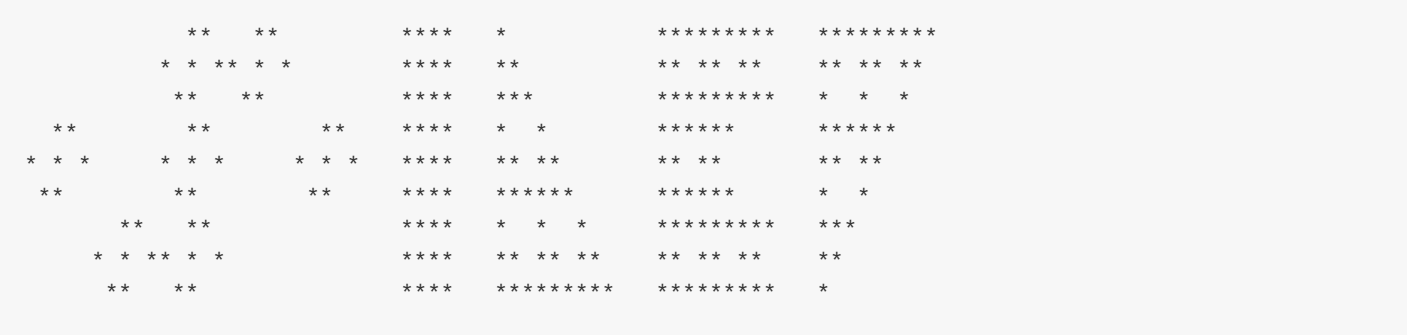
The Sidef Programming Language

Sidef is a modern, high-level, general-purpose programming language, inspired by Ruby, Raku and Go.



- The main features of Sidef include:
    - object-oriented programming
    - functional programming
    - functional pattern matching
    - optional lazy evaluation
    - multiple dispatch
    - lexical scoping
    - lexical closures
    - keyword arguments
    - regular expressions
    - support for metaprogramming
    - support for using Perl modules
    - optional dynamic type checking
    - big integers, rationals, floats and complex numbers

WWW

- Github: https://github.com/trizen/sidef
- Gitbook: https://trizen.gitbook.io/sidef-lang/ (legacy)
- RosettaCode: https://rosettacode.org/wiki/Sidef

LICENSE AND COPYRIGHT

import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package. If you institute patent litigation (including a cross-claim or counterclaim) against any party alleging that the Package constitutes direct or contributory patent infringement, then this Artistic License to you shall terminate on the date that such litigation is filed.

Disclaimer of Warranty: THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Preface

In this book, we are going to take a complete tour of the Sidef programming language; a modern language which explores the richness of the object-oriented paradigm and the elegant ideas of functional programming.

All started out as an idea in the early Spring of 2013 when I started discussing with a good friend of mine, Ioana Fălcușan, about the concept of a new programming language.

The original design described a simple and elegant object-oriented programming language, that was very easy to implement in terms of writing a parser and an interpreter for it.

About a week later, in March 30, 2013, we started the Sidef programming language project on GitHub. By the way, *Sidef* (pronounced *C-def*) is a Romanian word which means *nacre* in English.

As the time was passing by, we eventually started moving from AST interpretation towards code generation. The language now features a full code generator that walks the AST of a Sidef program, generating equivalent Perl 5 code. This method of execution made the implementation several times faster and also allowed the addition of more language constructs.

At the time of writing this book, Sidef is a little bit more than 5 years old and it vigorously continues to evolve with each monthly (or so) release, taking the best from other modern languages, like Perl 6, Julia and Ruby.

In the second half of the book, the reader can find a large collection of programming tasks implemented in Sidef, illustrating various ways for how the language can be used.

WWW

- Github: https://github.com/trizen
- Blogspot: https://trizenx.blogspot.com

Email

If you have any questions or find any issues, please contact me at `trizen@cpan.org`.

Getting started

Instructions for downloading and installing the Sidef interpreter.

Installation

Sidef can be installed automatically from the CPAN, by invoking the following command:

```
$ cpan Sidef
```

If the testing takes a long time, add the `-T` flag to build and install Sidef without testing:

```
$ cpan -T Sidef
```

When the `cpan` command is not available, try:

```
$ perl -MCPAN -e "CPAN::Shell->install(q{Sidef})"
```

IMPORTANT: Sidef needs the GMP, MPFR and MPC C libraries.

Installing from git source

To install Sidef manually, download the latest version, unzip it and follow the installation steps:

```
$ perl Build.PL
# ./Build installdeps
# ./Build install
```

When Module::Build is not installed, try:

```
$ perl Makefile.PL
$ make test
# make install
```

Linux installation

Arch Linux

Sidef is available on the AUR and can be installed using an AUR helper, like trizen:

```
$ trizen -S sidef
```

Debian / Ubuntu / Linux Mint

On Debian-based distributions, Sidef can be installed from the CPAN, by executing the following commands:

```
$ sudo apt install libgmp-dev libmpfr-dev libmpc-dev libc-dev cpanminus
$ sudo cpanm -n Sidef
```

Android installation

It's also possible to install Sidef on Android, by installing Termux and executing the following commands:

```
$ pkg install perl make clang libgmp libmpfr libmpc
$ cpan -T Sidef
```

If the installation succeeded, the `sidef` command should be available:

```
$ sidef -h
```

Running Sidef without installation

It is also possible to run Sidef without having to install it. If you're using a Unix-like OS, all you need to do is to execute the following commands:

```
$ wget 'https://github.com/trizen/sidef/archive/master.zip' -O 'master.zip'
$ unzip 'master.zip'
$ cd 'sidef-master/bin/'
$ ./sidef -v
```

Those commands will download and unpack the latest version of Sidef and will execute the `bin/sidef` script which will print out the current version of the language.

To execute a Sidef script, run:

```
$ ./sidef ../scripts/sierpinski_triangle.sf
```

You can, also, add the following alias to your `~/.bashrc` or `~/.zshrc`:

```
alias sidef="/path/to/bin/sidef"
```

Creating the first Sidef script

A Sidef script can be written in any text editor and, by convention, it has the **`.sf`** extension.

The content of a simple *Hello World* program looks like this:

```
#!/usr/bin/sidef

say "Hello, 世界";
```

If we save the content in a new file called **`hello.sf`**, we can execute the code by running:

```
$ sidef hello.sf
```

Sidef interpreter

This section describes the details of the main Sidef interpreter

Command-line options

The Sidef interpreter has the following command-line options:

```
Usage: sidef [switches] [--] [programfile] [arguments]

  -c            compile the code into a Perl program
  -C            check syntax only
  -D            dump the syntax tree of a program
  -E program    one line of program
  -H            interactive help
  -i            interactive mode
  -k            keep track of potential unsafe parser interpretations
  -M mode       set the rounding mode of floating-point numbers
                valid modes: [near], zero, inf, +inf, -inf
  -o file       file where to dump the output
  -O level      perform code optimizations before execution
                valid levels: [0], 1, 2
  -P int        set the precision of floating-point numbers (default: 48)
  -r            parse and deparse a Sidef program
  -R lang       parse and deparse a Sidef program into a given language
                valid values: sidef, perl
  -s            save compiled code in a database to reduce boot-time
  -t            treat all command-line arguments as scripts
  -v            print version number and exit
  -w            enable warnings with stack backtrace
  -W            make warnings fatal (with stack backtrace)

Run 'sidef -i' for interactive mode.
```

...which we'll examine in more detail in the following pages.

One-line programs

The `-E code` command will execute the code specified as a command-line argument.

Example:

```
$ sidef -E "say 'hello world'"
```

Outputs:

```
hello world
```

Interactive mode

The interactive mode (a.k.a. REPL) is available by simply executing the `sidef` command, or by specifying the `-i` command-line switch.

```
$ sidef -i
Sidef 3.99, running on Linux, using Perl v5.34.0.
Type "help", "copyright" or "license" for more information.
>> n = 41
=> 41
>> n**2 + n - 1
=> 1721
>>
```

Parser warnings

Sidef provides the `-k` option which will keep track of all the possible unsafe parser interpretations.

For example, if we declare the following function, but we misspell its name when we call it, Sidef will interpret it as a method call, which is probably not what we want:

```
func foo(n) { say n }
fo(42)                  # will get interpreted as `42.fo`
```

When the command-line option `-k` is specified, the following warning is produced:

```
[INFO] `fo` is parsed as a prefix method-call at script.sf line 2
```

One simple way to catch this kind of errors at parse-time, it to always call functions with the explicit `.call` method:

```
func foo(n) { say n }
fo.call(42)
```

The above code will fail to compile, producing the following error message:

```
File : test.sf
Line : 2
Error: variable <fo> is not declared in the current scope

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
fo.call(42)
^
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[?] Did you mean: foo
```

Dumping the AST

The `-D` command-line option dumps the abstract syntax tree (AST) of a given Sidef program.

Example:

```
$ sidef -D script.sf      # will dump the AST of script.sf
```

This feature is used mainly in debugging the parser and it requires the Data::Dump Perl module.

Optimization

The `-O level` command-line option controls the level of optimization before the execution begins.

Currently, there are three level of optimization available:

```
0 -- Does nothing. (default)
1 -- Does constant folding on the AST. (recommended)
2 -- Does constant folding, after which it deparses the AST into Sidef code, parses the code again
and does more constant folding on the new AST.
```

In the end, the code is translated to Perl and is ready to be executed. In the translation process, however, more non-optional optimizations are performed, such as loop and scope optimizations.

Deparsing

Deparsing is the reverse process of parsing, which translates the AST back into code. Currently, Sidef supports deparsing into two languages with the `-R lang` command-line switch:

- `-R perl`
  - Deparses the AST into valid Perl code.
- `-R sidef`
  - Deparses the AST into valid Sidef code.

Example:

```
$ sidef -Rperl script.sf | perl
```

The `-Rsidef` switch (or simply `-r`) is useful for verifying how the code is parsed.

Example:

```
$ sidef -r -E '1 + 2/3'
```

Outputs:

```
(1)->+((2)->/(3));
```

Deparsing can also be enabled in interactive mode:

```
$ sidef -i -r
Sidef 3.05 on linux
Type "help", "copyright" or "license" for more information.
>> [1,2,3].map { .sqrt }
[1, 2, 3]->map({|_| (_->sqrt()) });
>>
```

Precompilation

Sidef supports precompilation by saving each compiled code inside a database, which is updated automatically and sanitized periodically.

This method reduces significantly the boot-time of Sidef programs, and it works as following:

- it checks the database with the MD5 of the code
- if the MD5 exists inside the database, it returns the executable code

otherwise:

- parses the code and generates the executable code
- stores the executable code inside the database with the MD5 of the code

Next time when the same code is executed, Sidef will simply retrieve the executable code from the database, without generating it again:

```
$ sidef -s script.sf             # may load slow the first time
$ sidef -s script.sf             # will load much faster the second time
```

When a given code is not executed again in the next two or three days, it will be removed automatically from the database. The code which is used frequently, will not be removed.

Compilation

A Sidef script can be compiled to a stand-alone Perl program using the `-c` command-line option:

```
$ sidef -o out.pl -c script.sf
```

The above command will compile the file `script.sf` into the Perl script `out.pl`, which will include the entire implementation code of Sidef.

Currently, Sidef code that contains `eval()` cannot be compiled correctly to Perl, as it requires some parse-time information for run-time evaluation, which is lost in the compilation process.

Syntax and semantics

This section describes the pieces of the language and their meanings in great detail.

Comments

Single-line comments

The sharp (`#`) character sequence marks the following text as a single-line comment. Single-line comments end at the end-of-line.

```
#
## The hello-world program...
#
say "Hello, world!";         # with single-line comments.
```

Multiple-line comments

Comments can span multiple lines by using the multiple-line comment style. Such comments start with `/*` and end with `*/`. The text between those multi-line comment markers is the comment.

```
/*
   This is another style of a comment.
   It allows multiple lines.
*/
```

Embedded comments

This kind of comments are a little bit weird at the first glance, but they are useful sometimes.

```
var speed = (distance #`(in meters) / time #`(in seconds));
```

is equivalent with:

```
var speed = (distance / time);
```

A common use of embedded comments is to specify a shell evaluation code to execute the actual Sidef script when it's executed by a shell program. However, the eval statement is completely ignored by Sidef since the block in which it is defined, it's never executed.

```
#`(if running under some shell) {
    eval 'exec /usr/bin/sidef $0 ${1+"$@"}'
}
```

Built-in types

Several built-in types are available for creating new objects.

Nil

The `nil` type represents the absence of a value, and is considered undefined:

```
nil
defined(nil)  # -> false
nil.class     # -> error, methods cannot be called on undefined values
nil.ref       # -> error, methods cannot be called on undefined values
```

The `nil` type has only one possible representation, `nil`.

Instead of an error or exception, an *undefined* value is represented by `nil`:

```
Hash(){:key}    # -> nil
```

Bool

Bool has two possible values: `true` and `false`:

```
true     # A Boolean value that is true
false    # A Boolean value that is false
```

Alternatively, one can write:

```
Bool(1)     # true
Bool(0)     # false
```

Numbers

```
# Integers
say factorial(30)              #=> 265252859812191058636308480000000
say fibonacci(164)             #=> 840403783297413488274367626780173

# Floating-point numbers
say sqrt(Num.pi)               #=> 1.772453850905516027298167483 3[...]
say exp(1234)                  #=> 8.30597593736179421825852129 13[...]e535

# Rational numbers
var x = 2/3
say x*3                        #=> 2
say 2/x                        #=> 3
say x.as_frac                  #=> 2/3

# Complex numbers
say Complex(3, 4)              #=> 3+4i
say 3+4.i                      #=> 3+4i
say sqrt(-4)                   #=> 2i
say log(-1)                    #=> 3.14159265358979323846264338 32[...]i
```

The `Number` type (also aliased as `Num`) represents any Number object (including complex numbers):

```
say 42.kind_of(Number)          #=> true
say sqrt(-1).kind_of(Number)    #=> true
```

Additionally, the `Number` class can be used for constructing new Number objects:

```
Number("1234.52")          # set decimal number
Number("10110111", 2)      # set binary number
Number("deadbeef", 16)     # set hexadecimal number
```

Integer literals

Sidef supports arbitarily large integers, by using Math::GMPz, which is the Perl interface to the GMP library.

```
255              # decimal
0xff             # hexadecimal
0377             # octal
0b1111_1111      # binary
```

A numerical string (in any base from 2 to 62) can be converted into a number as following:

```
Number("1234")          # from string in base 10 (default)
Number("foo", 36)       # from string in base 36
```

The `Number.base(b)` method provides conversion from numbers into strings in a given base:

```
1234.base(13)       # to string in base 13
1234.base(36)       # to string in base 36
```

Rational

Rational numbers are built-in, using Math::GMPq, which provides a Perl interface to the `mpq` layer in the GMP library.

```
3/4         # rational value: 3/4
2.5         # parsed and represented in rational form: 5/2
```

A numerical string can be converted into a rational number by using the `Number` class:

```
Number("0.75")          # "0.75" is parsed and stored in rational form as 3/4
Number("fff/aaa", 36)   # parse a base-36 fraction as 4095/2730
```

The Number method `as_rat` can be used for getting the rational form of a number:

```
say 3/4               # 0.75
say as_rat(3/4)       # 3/4
say as_rat(1.234, 36) # h5/dw   (which is 617/500 in base-10)
```

Float

In Sidef, all literal numbers are specially parsed and converted implicitly into a rational number, which are internally represented by Math::GMPz and Math::GMPq.

```
1.234           # 1.234
.1234           # 0.1234
1234e-5         # 0.01234
12.34e5         # 1234000
```

This means that Sidef, by default, does not suffer from the floating-point inaccuracy:

```
0.1 + 0.2 == 0.3    # true
```

An integer or a rational number can be converted to a floating-point value by using the `.float` Number method, which

represents floating-points using Math::MPFR (which is the Perl interface to the MPFR library) at a default precision of 192 bits:

```
1.234.float         # conversion to floating-point number
```

In numerical operations, a floating-point number will automatically propagate:

```
var x = 3/4         # rational
var y = 0.9.float   # floating-point
var z = x+y         # floating-point
```

The precision of floating-point numbers can be changed by modifying the `Num!PREC` class-variable:

```
say sqrt(2)          # 1.41421356237309504880168872420969807856967187538
local Num!PREC = 24  # number of bits of precision
say sqrt(2)          # 1.41421
```

By using the `local` keyword, the precision will be changed only locally (which also includes any function/method call in the current scope):

```
func my_sqrt(n) {
    sqrt(n)
}

say my_sqrt(2)              # 1.41421356237309504880168872420969807856967187538

do {                        # creates a new scope
    local Num!PREC = 24     # changes the floating-point precision locally
    say my_sqrt(2)          # 1.41421
}                           # the default precision is restored when the scope ends

say my_sqrt(2)              # 1.41421356237309504880168872420969807856967187538
```

Complex

Support for complex numbers is provided by Math::MPC, which is a Perl interface to the MPC library.

```
Complex(3, 4)
```

Alternatively:

```
3 + 4i
3:4
```

Complex numbers are deeply integrated into the language and can be used in combination with all the other Number types (with implicit propagation):

```
sqrt(-1)        # 1i
log(-1)         # 3.14159265358979323846264338327950288419716939938i
4 + sqrt(-1)    # 4+i
(3+4i)**2       # -7+24i
```

All complex numbers are Number objects:

```
(3 + 4i).class       # Number
Complex(3, 4).class  # =//=
```

Fraction

The Fraction class represents a generic fraction:

```
var a = Fraction(3, 4)
var b = Fraction(5, 7)

say a*b     #=> Fraction(15, 28)
say a+b     #=> Fraction(41, 28)
```

Mod

This class represents a modular operation, similar to PARI/GP's built-in `Mod(a,m)` class.

```
Mod(3, 4)       # represents 3 mod 4
```

Example:

```
var a = Mod(13, 19)

a += 15             # Mod(9, 19)
a *= 99             # Mod(17, 19)
a /= 17             # Mod(1, 19)

say a               # Mod(1, 19)
say (a == 1)        # true
say (a == 20)       # true

a -= 43             # Mod(15, 19)

say a**42           # Mod(11, 19)
say a**(-1)         # Mod(14, 19)
say sqrt(a+1)       # Mod(4, 19)

say chinese(Mod(43, 19), Mod(13, 41))   # Mod(423, 779)
```

Gauss

The `Gauss` class provides support for rational Gaussian numbers and various operations on these numbers.

```
Gauss(3, 4)     # represents 3+4i
```

Example:

```
say Gauss(3,4)**100
say Mod(Gauss(3,4), 1000001)**100   #=> Mod(Gauss(826585, 77265), 1000001)

var a = Gauss(17,19)
var b = Gauss(43,97)

say a+b     #=> Gauss(60, 116)
say a-b     #=> Gauss(-26, -78)
say a*b     #=> Gauss(-1112, 2466)
say a/b     #=> Gauss(99/433, -32/433)
```

Quadratic

The `Quadratic` class represents a quadratic integer of the form: `a + b*sqrt(w)`.

```
var x = Quadratic(3, 4, 5)  # represents: 3 + 4*sqrt(5)
var y = Quadratic(6, 1, 2)  # represents: 6 + sqrt(2)

say x**10       #=> Quadratic(29578174649, 13203129720, 5)
say y**10       #=> Quadratic(253025888, 176008128, 2)

say x.powmod(100, 97)   #=> Quadratic(83, 42, 5)
say y.powmod(100, 97)   #=> Quadratic(83, 39, 2)
```

Quaternion

The `Quaternion` class represents a quaternion number of the form `a + b*i + c*j + d*k`, where `a`, `b`, `c`, and `d` are real numbers; and `i`, `j`, and `k` are the basic quaternions.

```
var a = Quaternion(1,2,3,4)
var b = Quaternion(5,6,7,8)

say a+b     #=> Quaternion(6, 8, 10, 12)
say a-b     #=> Quaternion(-4, -4, -4, -4)
say a*b     #=> Quaternion(-60, 12, 30, 24)
say b*a     #=> Quaternion(-60, 20, 14, 32)
say a/b     #=> Quaternion(35/87, 4/87, 0, 8/87)

say a**5                #=> Quaternion(3916, 1112, 1668, 2224)
say a.powmod(43, 97)    #=> Quaternion(61, 38, 57, 76)
say a.powmod(-5, 43)    #=> Quaternion(11, 22, 33, 1)
```

Polynomial

The `Polynomial` class implements support for polynomials.

```
say Polynomial(5)                   # monomial: x^5
say Polynomial([1,2,3,4])           # x^3 + 2*x^2 + 3*x + 4
say Polynomial(5 => 3, 2 => 10)     # 3*x^5 + 10*x^2
```

Also aliased as `Poly()`:

```
var a = Poly([1,2,3])
var b = Poly([4,5,-6,7])

say a+b     #=> 4*x^3 + 6*x^2 - 4*x + 10
say a-b     #=> -4*x^3 - 4*x^2 + 8*x - 4
say a*b     #=> 4*x^5 + 13*x^4 + 16*x^3 + 10*x^2 - 4*x + 21

say 42-a    #=> -x^2 - 2*x + 39
say 42+b    #=> 4*x^3 + 5*x^2 - 6*x + 49
say 42*b    #=> 168*x^3 + 210*x^2 - 252*x + 294

say a/42    #=> 1/42*x^2 + 1/21*x + 1/14
say b/42    #=> 2/21*x^3 + 5/42*x^2 - 1/7*x + 1/6
```

String

A String represents an immutable sequence of UTF-8 characters.

Double quoted strings

A String object is typically created with a string literal, enclosing UTF-8 characters in double quotes:

```
"hello world"
```

Alternatively, we can create strings using the *String* class:

```
String("hello world")
```

Being a new programming language, Sidef has also built-in support for Unicode quotation marks:

```
„double quoted”        # == "double quoted"
```

A backslash can be used to denote some characters inside the string:

```
"\""    # double quote
"\\"    # backslash
"\e"    # escape
"\f"    # form feed
"\n"    # newline
"\r"    # carriage return
"\t"    # tab
"\s"    # space
"\v"    # vertical tab
```

One can use `\o{...}` to denote a code point written in octal:

```
"\o{101}"   # == "A"
"\o{123}"   # == "S"
"\o{12}"    # == "\n"
"\o{1}"     # string with one character with code point 1
```

Or `\x{...}` and specify hexadecimal numbers:

```
"\x{41}"    # == "A"
"\x{263a}"  # == "☺"
```

To specity Unicode names, one can use `\N{...}` :

```
"\N{WHITE SMILING FACE}"            # == "☺"
"\N{GREEK CAPITAL LETTER GAMMA}"    # == "Γ"
```

A string can span multiple lines:

```
"hello
world"    # same as "hello\nworld"
```

For writing a string that has many double quotes, parenthesis, or similar characters, one can use alternative literals:

```
# Supports double quotes and nested parenthesis
%(hello ("world"))  # same as "hello (\"world\")"

# Supports double quotes and nested brackets
%[hello ["world"]]  # same as "hello [\"world\"]"

# Supports double quotes and nested curlies
%{hello {"world"}}  # same as "hello {\"world\"}"

# Supports double quotes and nested angles
%<hello <"world">>  # same as "hello <\"world\">"
```

The Parser is aware of Unicode delimiters as well. Here are only a few examples:

```
%„...”
%«...»
%⟨...⟩
%⌈...⌋
```

Interpolation

Creating a String with embedded expressions, is called string interpolation:

```
"sum = #{1 + 2}"        # "sum = 3"
```

Single quoted strings

Single quoted strings does not support escapes, nor interpolation.

```
'single\tquoted'   # creates a string as it is
```

For specifying a custom delimiter, one can use `%q` followed by any non-whitespace delimiter:

```
%q(hello ('world'))     # same as 'hello (\'world\')'
```

Another way of writing string literals, is by placing a colon in font of an alphanumeric string that begins with a letter.

```
:word          # == 'word'
:another_word  # == 'another_word'
```

Here-document

There must not be a space between the `<<` and the token string. When the token string is double-quoted (`""`) or not quoted, the content will be interpolated like a double-quoted string:

```
<<"EOF";
a = #{1+2}
b = #{3+4}
EOF
```

If single quotes are used, then the here document will not support interpolation, like a normal single-quoted string:

```
<<'FOO';
No
#{interpolation}
here
FOO
```

The here document does not start immediately at the `<<END` token -- it starts on the next line. The `<<END` is actually an expression, whose value will be substituted by the contents of the here document. To further illustrate this fact, we can use the `<<END` inside a complex, nested expression:

```
(<<EOF + "lamb");
Mary had
  a little
EOF
```

which is equivalent with:

```
(<<EOF
Mary had
  a little
EOF
+ "lamb");
```

Regex

A Regex object represents a regular expression, which is fully Perl5 compatible.

```
/^my?\s*re(g|[ex])\z/i
```

Alternatively:

```
Regex('^my?\s*re(g|[ex])\z', 'i')
```

File

A File object represents the name of a local file:

```
File("/path/to/file.ext")
```

Alternatively:

```
%f(/path/to/file.ext)
%d(/path/to) + %f(file.ext)
```

A file can be opened for reading or writing, using the `open_r` and `open_w` methods, respectively:

```
var file = File("file.txt")    # File object
var fh = file.open_r           # FileHandle object
say fh.lines                   # read the lines into an Array
```

Creating a new file:

```
var fh = File("newfile.txt").open_w
fh.say("foo")   # write a newline to the file
fh.close        # close the filehandle
```

A File abstracts an entity on the filesystem by name; a FileHandle abstracts a file descriptor in memory.

```
File("file.txt").read           # read the content of a file as a String (UTF-8)
File("file.txt").read(:raw)     # read the content of a file as a String (RAW)
```

See the FileHandle documentation for more information.

FileHandle

Abstraction on a file descriptor, to interact with the content of a file.

```
var a = File("existing_file.txt").open_r      # open file for reading
var b = File("new_file.txt").open_w           # open file for writing
```

Some useful methods on FileHandle objects:

```
fh.autoflush(bool)      # enable/disable auto-flush mode
fh.binmode(encoding)    # set encoding
fh.line                 # read one line as a String
fh.slurp                # read the entire file as a String
fh.lines                # read the entire file as an Array of String objects
fh.grep { ... }         # collect only the lines that match the block
fh.grep(/regex/)        # collect only the lines that match the regex
fh.each { ... }         # iterate over each line
fh.eof                  # true when at the end of the file
fh.tell                 # current position in the file
fh.seek(pos,whence)     # jump to this position in the file
fh.lock                 # lock the filehandle
fh.unlock               # unlock the filehandle
fh.say(str)             # write a string into the file (appending "\n")
fh.print(str)           # write a string into the file (without appending "\n")
fh.close                # close the filehandle
```

The special `FileHandle` type can be for checking if a given object is really a FileHandle object:

```
fh.kind_of(FileHandle)    # true if `fh` is a FileHandle object
```

The file object to which a FileHandle refers, if any, can be obtained with `fh.file`, which may be `nil` or a `File` object.

Dir

The Dir object represents the name of a local directory:

```
Dir("/my/path")
```

Alternatively:

```
%d(/my/path)
%d(/my)+%d(path)
```

See the DirHandle documentation for more information.

DirHandle

Abstraction on a directory descriptor, to access the content of a directory:

```
var dh = Dir("/my/path").open
```

Some useful methods on DirHandle objects:

```
dh.read                 # read one entry as a File or a Dir object
dh.entries              # read all entries as an Array of File and Dir objects
dh.files                # read all file entries as an Array of File objects
dh.dirs                 # read all directory entries as an Array of Dir objects
dh.seek(pos)            # set the current position in the directory handle
dh.rewind               # set the current position to the beginning of the directory
dh.tell                 # returns the current position in the directory handle
dh.chdir                # change the working directory to the path of `dh`
dh.each { ... }         # iterate over each entry
dh.close                # close the directory handle
```

The special `DirHandle` type can be for checking if a given object is really a DirHandle object:

```
dh.kind_of(DirHandle)    # true if `dh` is a DirHandle object
```

The directory object to which a DirHandle refers, if any, can be obtained with `dh.dir`, which may be `nil` or a `Dir` object.

Array

An Array is a collection of objects which can grow or shrink dynamically.

```
[123, "abc", true, nil]
```

Alternatively, one can write:

```
Array(123, "abc", true, nil)
```

Working with arrays

Elements of an array can be accessed with the special syntax `array[i]` where `i` is a zero-based index inside the array.

```
var array = [1, 2, 3, 4, 5]

array[0] = 6
array[1] = 7

say array
```

Special arrays

Arrays of strings or numbers can be created with a special syntax:

```
%w(1 two three)         # same as: ["1", "two", "three"]
%n(1 2.5 3.75)          # same as: [1, 2.5, 3.75]
%i(1 2.5 3.75)          # same as: [1, 2, 3]
```

Unescaped spaces and unescaped comments are removed. For example, the following declaration:

```
%w(
    Sidney        # Australia
    New\ York     # U.S.A
)
```

...is equivalent with `["Sidney", "New York"]`.

There is also `%W(...)` which understands escapes and interpolation:

```
%W(
    hello\tworld
    one\ item         # some comment
    \#{1+2}
)
```

...which means: `["hello\tworld", "one item", "3"]`.

Another way is by using the `<...>` and `«...»` delimiters:

```
<I 8 some 3.14>         # same as: ["I", "8", "some", "3.14"]
```

Pair

The Pair class represents a 2-element array.

```
var p = Pair(42, 99)

say p.first      #=> 42
say p.second     #=> 99

p.first = "foo"     # change first element
p.second = "bar"    # change second element

say p       #=> Pair("foo", "bar")
```

Linked lists can be created using nested Pair objects:

```
var ll = Pair(1, Pair(2, Pair(3, 4)))

loop {
    say ll.first
    if (ll.second.kind_of(Pair)) {
        ll = ll.second
    }
    else {
        say "Reached end: #{ll.second}"
        break
    }
}
```

NamedParam

NamedParam is the type used to give named parameters to callables.

Its literal syntax uses the infix `:` operator, which may have an empty right-side, but whose left-side must not be a literal, and may be a name in the current scope.

```
var a = "b"
"a": "b"                  # -> Pair("a", "b")
(a : "b")                 # -> Pair("b", "b")
a: "b"                    # -> NamedParam a: "b"

func foo(a, b = 2) { a/b }

var p = a: 4
say foo(p)                # -> 2

var args = [b:7, a:21]
say foo(args...)          # -> 3
```

It is not possible to write a function which takes a literal `NamedParam` argument, but this can be overcome by passing a variable reference that holds a `NamedParam` object, and later dereferencing it.

```
func baz(param) {
  "Have #{param}, #{*param}"
}
var a = a:1
say baz(a)   # error: no matching function for parameter a
say baz(\a)  # prints: Have REF(0x55f9455ca270), a: 1
```

Vector

The `Vector` built-in class (which is a child of the `Array` class), provides support for defining and working with vectors:

```
Vector(1, 2, 3, 4, 5)
```

Alternatively:

```
%v(1 2 3 4 5)
```

Operations

```
var A = Vector(1, 2, 3, 4)
var B = Vector(9, 8, 7, 6)

say (A + B)                  # vector entrywise addition
say (A - B)                  # vector entrywise subtraction
say (A * B)                  # cross product

say (A + 42)                 # vector-scalar addition
say (A - 42)                 # vector-scalar subtraction
say (A * 42)                 # vector-scalar multiplication
say (A / 42)                 # vector-scalar division

say A.sum                    # vector sum
say A.prod                   # vector product

say A.norm                   # normalized value: sum of squares
say A.manhattan_norm         # Manhattan normalized: sum of absolute values

say A.dist(B)                # distance between two vectors
say A.dist_norm(B)           # distance squared between two vectors
say A.manhattan_dist(B)      # Manhattan distance between two vectors
say A.chebyshev_dist(B)      # Chebyshev distance between two vectors
```

Matrix

The `Matrix` built-in class (which is a child of the `Array` class), provides support for defining and working with matrices:

```
Matrix(
    [1, 2],
    [3, 4],
)
```

Alternatively:

```
%m(1 2; 3 4)
```

Operations

A subset of `Matrix` operations are included in the following example:

```
var A = Matrix(
    [2, -3,  1],
    [1, -2, -2],
    [3, -4,  1],
)

var B = Matrix(
    [9, -3, -2],
    [3, -1,  7],
    [2, -4, -8],
)

say (A + B)     # matrix addition
say (A - B)     # matrix subtraction
say (A * B)     # matrix multiplication
say (A / B)     # matrix division

say (A + 42)    # matrix-scalar addition
say (A - 42)    # matrix-scalar subtraction
say (A * 42)    # matrix-scalar multiplication
say (A / 42)    # matrix-scalar division

say A**20       # matrix exponentation
say A**-1       # matrix inverse: A^-1
say A**-2       # (A^2)^-1

say B.det             # matrix determinant
say B.solve([1,2,3])  # solve a system of linear equations
```

Hash

An Hash is a dynamic collection of key-value pairs. The keys must be of type String, while the values can have any type.

```
Hash(
    a => 1,
    b => 2,
    c => 3,
)
```

Working with hashes:

Elements of an hash can be accessed with the special syntax `hash{key}` where `key` is a String-convertable object.

```
var hash = Hash(name => 'Sidef')

# Print the value of a key
say hash{"name"}      #=> "Sidef"

# Set a key into the hash
hash{"age"} = 6

# Print the hash
say hash
```

Hash of Arrays

The idiom `hash{key} := [] << value` can be used for creating an hash of arrays, as illustrated in the example below:

```
var hash = Hash()

for p in (primes(100)) {
    for d in (divisors(p-1)) {
        if (powmod(2, d, p) == 1) {
            hash{d} := [] << p
        }
    }
}

say hash.grep_v { .len > 1 }
```

which outputs:

```
Hash(
    "10" => [11, 31],
    "11" => [23, 89],
    "18" => [19, 73],
    "22" => [23, 89],
    "36" => [37, 73]
)
```

Existent key

A key can be checked if it exists in a hash using the syntax: `hash.has(key)`.

Deleting a key

A key can be deleted from a hash using the syntax: `hash.delete(key)`.

Set

A set is an ordered collection of objects, with no duplicates.

```
Set('foo', 'bar', 'baz')
```

Operations

All the set operators, such as intersection, difference, symmetric difference, union and concatenation, are supported.

```
var A = Set('foo', 'bar', 'baz', 'foo')
var B = Set('bar', 'foo', 'qux')

# Intersection
say (A & B)         #=> Set("foo", "bar")

# Union
say (A | B)         #=> Set("foo", "qux", "bar", "baz")

# Difference
say (A - B)         #=> Set("baz")
say (B - A)         #=> Set("qux")

# Symmetric difference
say (A ^ B)         #=> Set("baz", "qux")

# Concatenation
say (A + B)         #=> Set("baz", "bar", "qux", "foo")
```

Updating

The method `set.delete(obj)` can be used for removing a given object from the set.

Bag

A bag (also known as a multi-set) is a unordered collection of objects, similar to a hash table, where each object has a count number, which represents the number of times it exists in the bag.

```
Bag('foo', 'bar', 'baz')
```

Operations

The Bag class supports all the set operators, such as intersection, difference, symmetric difference, union and concatenation.

```
var A = Bag('foo', 'bar', 'baz', 'foo')
var B = Bag('bar', 'foo', 'qux')

# Count how many times is 'foo' present in the bag A
say A.count('foo')  #=> 2

# Intersection
say (A & B)         #=> Bag("foo", "bar")

# Union
say (A | B)         #=> Bag("qux", "bar", "baz", "foo", "foo")

# Difference
say (A - B)         #=> Bag("baz", "foo")
say (B - A)         #=> Bag("qux")

# Symmetric difference
say (A ^ B)         #=> Bag("foo", "qux", "baz")

# Concatenation
say (A + B)         #=> Bag("foo", "foo", "foo", "bar", "bar", "baz", "qux")
```

Updating the bag

The methods `bag.add_pair(obj, count)` and `bag.update_pair(obj, count)` can be used for efficiently updating a bag in-place.

```
var A = Bag('foo', 'foo', 'bar')

# Add 'bar' with count=2
A.add_pair('baz', 2)

say A   #=> Bag("baz", "baz", "bar", "foo", "foo")

# Update the count of 'foo'
A.update_pair('foo', 1)

say A   #=> Bag("baz", "baz", "bar", "foo")
```

Furthermore, the method `bag.delete(obj)` can be used for removing one occurrence of object `obj` from the bag, while the `.delete_all(obj)` can be used for removing all the occurrences of `obj` from the bag.

Block

A Block is a special object which encapsulates zero or more statements which can be executed at a later time. The block itself can be stored inside variables, arrays or passed as argument to functions.

```
{
    say "Hello world!"
}
```

The `run` method on `Block` objects is used to call the block with arguments.

```
{
    say "Hello #{_}!"
}.run("you")           # prints "Hello you!"
```

Time

Represents the UNIX Epoch time.

```
Time()        # seconds since the Epoch to now
Time(sec)     # seconds since the Epoch to sec
```

Example:

```
var t = Time()      # Time object representing current time

say t.sec           # get the number of seconds (as a Number) from object t

say Time.micro      # micro-seconds since the Epoch (as a Number)
say Time.sec        # seconds since the Epoch (as a Number)

say t.local         # Date object with local time zone
say t.gmt           # Date object with Greenwich time zone
```

Date

The Date class provides support for working with dates.

```
var d1 = Date() # localtime date

say d1.day  #=> 5
say d1.mon  #=> 2
say d1.year #=> 2020

# Parse date
var d2 = Date.parse("2020-02-02", "%Y-%m-%d")

say (d1 - d2)       # difference in seconds
say (d1 <=> d2)     # date comparison

say d1.add_days(3)      # add n days
say d1.add_months(5)    # add n months
say d1.add_years(9)     # add n years

say (d1 == d2)      # check equality
say (d1 != d2)      # check inequality

say d1.epoch        # seconds since the epoch

# Format
say Time().local.format("%Y-%m-%d")        # current date

# Validate date
say Date.valid("2020-06-30", "%Y-%m-%d")    #=> true
say Date.valid("2020-06-31", "%Y-%m-%d")    #=> false
```

Sys

The Sys class provides low-level access to various system functions.

```
Sys.run("cmd")        # execute a command
Sys.osname            # name of operating system
Sys.sidef             # path to sidef executable

Sys.kill(:KILL, pid)  # send a "KILL" signal to pid
Sys.fork              # fork the self program
Sys.wait              # wait for a child process to finish

Sys.alarm(n)          # set alarm for n seconds
Sys.sleep(3.5)        # sleep 3.5 seconds
Sys.exit(2)           # exit the program with code 2

Sys.read(TYPE)        # read a type of data from STDIN
Sys.read(msg, TYPE)   # read a type of data from STDIN, with prompt
Sys.scanln(msg)       # read a String line from STDING, with prompt

Sys.refaddr(obj)      # internal reference address of an object
Sys.reftype(obj)      # internal reference type name of an object

Sys.eval(code)        # evaluate arbitrary Perl code
```

Language constructs

Several different constructs are available for achieving the most basic functionality as a programming language.

if

The `if` statement is one of the most basic conditional constructs.

```
var a = 21
var b = 42

if (a > b) {
    say "a is greater"
}
elsif (a == b) {
    say "a and b are equal"
}
else {
    say "b is greater"
}
```

A somewhat special feature is the support for capturing the original value evaluated in the `if` statement, such as:

```
if (some_function()) { |value|
    say value
}
```

Because Sidef is weakly typed, the condition inside the `if` statement can be any object, which is implicitly converted to a

`Bool` object. However, the value stored inside the captured block variable will be the result of the original expression, which may not necessarily be a `Bool` object.

Note that the captured value won't have any other methods called on it before being captured, for example the following may be unexpected:

```
var str = "ab"
if (str =~ /^a(.)$/) { |m|
    say m
    say m=="b"
    say m.class
    say m.dump
}
```

It will output

```
b
false
Match
(str =~ /^a(.)$/) # a Match object
```

because the entire Match object from the match expression `=~` will be captured by the Block. To save the first capture and make `m=="b"`, use `(str=~/^a(.)$/)[0]`.

with

The `with` statement behaves almost like the `if` statement, but instead of testing for trueness, it checks to see if the given argument is not a `nil` value.

```
with (obj) {

}
orwith (obj) {

}
else {

}
```

Same as the `if` statement, it supports capturing of a defined value in a block variable:

```
with (some_function()) { |value|
    say value
}
```

goto

The `goto` statement is the most basic form of unconditional transfer of control.

```
var i = 0

@:INCR i += 1
say "#{i} squared = #{i * i}"
goto :INCR if (i < 10)

say "Program Completed."
```

Output

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
5 squared = 25
6 squared = 36
7 squared = 49
8 squared = 64
9 squared = 81
10 squared = 100
Program Completed.
```

However, the goto statements have been considered harmful by many computer scientists, notably Dijkstra, so try to avoid using it if possible.

while

The `while` construct is almost like the `if` construct, except that it will keep executing its block as long the given expression is true.

```
var n = 10

while (n > 0) {
    say n
    n -= 1
}
```

do/while

A `do/while` construct guarantees the execution of the block at least once.

```
var name = ''

do {
    name = read("username: ", String)
} while(name ~~ /^(root|)\z/)

say "Your username is #{name}"
```

for

The `for` construct it's usually used for iteration over collections and for counting.

The following loop counts from 0 to 9:

```
for (var i = 0; i < 10; i++) {
    say i
}
```

Alternatively, using a simpler form of the for-loop:

```
for (0 ..^ 10) { |i|
    say i
}
```

To iterate over an array, Sidef supports yet another form of the `for` loop:

```
for i in [1,2,3,4] {
    say i
}
```

The `for-in` loop is a generalization of the `for` loop, supporting iteration over any object that respond to the `iter` method, which must return a block as the iterator, and the block should return one value at a time when it's called, or `nil` when there are no other values to return, which will break the loop.

Alternatively, the object may also respond to the `to_a` method, which must return an object of type `Array`. Using this knowledge, we can iterate over `Hash` objects because the `Hash` class implements the `to_a` method:

```
for i,k in Hash(a => 1, b => 2) {
    say "<#{i}> <#{k}>"
}
```

The `for-in` loop can also be used in iterating over matrices:

```
for i,j,k in [
    %w(a b c),
    %w(d e f),
] {
    say "<#{i}> <#{j}> <#{k}>"
}
```

loop

An infinite loop can be created using the `loop` keyword:

```
loop {
    say "Hello, world!"
}
```

gather/take

The `gather/take` construct was borrowed from Perl 6 and it's an interesting one.

```
var arr = gather {
    take(1)
    take(2)
    take(3)
}
say arr             # prints: [1,2,3]
```

Inside a `gather{}` block, the `take` keyword is available, which accepts a list of arguments that are stored inside an automatically created array. In the end, the gather returns the array to the caller.

given/when

The `given/when` construct is borrowed from Perl 6 and it's somewhat equivalent with `switch/case` from other languages.

```
given (42) {
  case (_ < 0) {
       say "Negative value!"
  }
  when (0) {
       say "Null value!"
  }
  when (1) {
       say "Value is: 1"
  }
  default {
       say "Value is greater than 1!"
  }
}
```

- `case` is used to test an expression for trueness
- `when` is used to test a value using the rules of the smartmarch operator

The rules of the smartmarch operator are pretty simple:

- `Obj ~~ Obj` will always check for equality when both objects have the same type.
- `Obj ~~ Array` will return true if `Obj` exists inside the array.
- `Obj ~~ Hash` will return true when `Obj` is a key inside the hash.
- `Obj ~~ Regex` will try to match the `Obj` against the regex and returns true on a successful match.

try/catch

A `try/catch` construct it's used to try an unsafe block of code and catch any unexpected errors.

```
var value = try   { isqrt(1764) }
            catch { die "error" }
say value   #=> 42
```

The `catch` branch is called only in case of a fatal error or when a `die` statement is executed in the `try` branch:

```
var value = try {
    "foo".some_undefined_method
} catch { |msg|
    say "Catched: #{msg}"
    42
}
say value     #=> 42
```

The `try/catch` construct returns the result of the last statement from the first successfully executed branch.

break

The `break` statement is used to exit early from loops.

```
for n in (1..100) {
    if (n.divisors.sum == 2*n) {
        say "#{n} is the first perfect number"
        break
    }
}
```

The statement can be used in any form of loops, including `while` and conditional `for` loops.

```
var i = 0
while (true) {
    if (i >= 10) { break }
    say ++i
}
```

next

The `next` statement skips one iteration of a loop.

```
for (var i = 0; i <= 5; i++) {
    if (i == 3) { next }
    say i
}
```

It can be used in all kinds of loops.

continue

The `continue` statement can be used inside a `given/when` construct to continue to the next branch.

```
given (10) { |i|
    when (10) {
        say "It's ten"
        continue
    }
    case (i.is_even) {
        say "It's even"
        continue
    }
    case (i % 5 == 0) {
        say "It's divisible by 5"
    }
}
```

Output:

```
It's ten
It's even
It's divisible by 5
```

Methods

A method is a distinct function defined for a specific type of object. There may be methods that share the same name, but each pointing at different functions, depending on the type of the object on which they are invoked.

```
"string".length     # String.length()
[1,2,3].length      # Array.length()
```

The methods can also be invoked using the prefix notation:

```
length("string")
length([1,2,3])
```

The prefix and postfix notations can be used interchangeably:

```
log("string".length)    # means: "string".length.log
length("string").log    # =//=
log(length("string"))   # =//=
```

A method can be invoked using the prefix notation, even when a function with the same name is declared in the same scope. This is done by preceding the method with `::`, as illustrated below:

```
func sqrt(n) { "sqrt of #{n} is #{n.sqrt}" }

say   sqrt(42)     # calls the `sqrt` function defined above
say ::sqrt(42)     # calls the `Number.sqrt()` method
```

Additionally, any alphanumeric method name can be used as an infix operator, by surrounding it with two backticks:

```
(1 `add` 2)           # means: 1.add(2)
(Math `sum` (1,2,3))  # means: Math.sum(1,2,3)
```

There is also support for calling a method which its name is not known until at run-time, which can be any expression that evaluates to a string:

```
say ( 50.(['+', '-'].rand)(30) )    # prints 20 or 80
```

If a method is not found for a given object, Sidef will throw a run-time error.

Operator precedence

All operators have the same precedence, which is controlled by the lack of whitespace between the operands.

```
1+2 * 3+4   # means: (1+2) * (3+4)
```

In the above example, the lack of whitespace between `1`, `+` and `2`, classifies the operation as a distinct expression.

The implications are the following:

```
var n = 1 + 2       # incorrect -- it means: (var n = 1) + 2
var n = 1+2         # correct
var n = (1 + 2)     # correct
```

When no precedence is defined, the order of operations is from left to right:

```
1 + 2 * 3       # means: ((1 + 2) * 3)
```

On the other hand, when too much precedence is defined, the order is from right to left:

```
1+2*3           # means: (1 + (2 * 3))
```

The precedence can also be controlled by backslashing or preceding the operator with a dot.

```
1 + 2 \* 3      # means: (1 + (2 * 3))
1 + 2 .* 3      # =//=
```

Method precedence

Method precedence is controlled by the following two method separators: `.` and `->`.

```
1 + 25.sqrt      # means: 1 + sqrt(25)
1 + 25->sqrt     # means: sqrt(1 + 25)
```

The rules are the following:

- `.` binds the method to the object which precedes the dot
- `->` makes everything from its left-side an expression and applies the method on the result

The infix backslash (`\`) removes any leading or trailing whitespace at that current position and it's useful for expanding method calls on multiple lines:

```
say "abc".uc      \
         .reverse \
         .chars
```

is equivalent with:

```
say "abc".uc.reverse.chars
```

Metaoperators

In Sidef we have an interesting set of metaoperators, which provide an easier way of working with arrays and matrices.

Unroll operator

It's a nice metaoperator borrowed from Perl 6, which unrolls two arrays and applies the operator on each two element-wise objects, creating a new array with the results. The operator can be a method or any other valid operator and must be enclosed between `» «` or `>> <<`.

```
[1,2,3] »+« [4,5,6]           # [1+4, 2+5, 3+6]
%w(a b c) >>cmp<< %w(c b a)   # [-1, 0, 1]
```

Internally, the `unroll_operator` method is called, which can, also, be implemented in user-defined classes.

Map operator

The array map operator works exactly like the `Array.map{}` method, but it's slightly more efficient and easier to write. The map operator must be enclosed between `» »` or `>> >>`.

```
[1,2,3] »*» 4      # [1*4, 2*4, 3*4]
```

Internally, the `map_operator` method is called.

Pam operator

The pam operator is kind of a reversed mapping of the array ("pam" is "map" spelled backwards), where the provided argument is used as the first operand to the operator provided. The operator must be enclosed between `« «` or `<< <<`.

```
[1,2,3] «/« 10     # [10/1, 10/2, 10/3]
```

Internally, the `pam_operator` method is called.

Reduce operator

This metaoperator reduces an array to a single element. The operator needs to be enclosed inside `« »` or `<< >>`.

```
[1,2,3]«+»      # 1 + 2 + 3
[1,2,3]«/»      # 1 / 2 / 3
```

Internally, the `reduce_operator` method is called.

Cross operator

The metaoperator `~X` or `~Xop` crosses two arrays and returns a new one.

```
[1,2] ~X+ [3,4]    # [1+3, 1+4, 2+3, 2+4]
[1,2] ~X  [3,4]    # [[1,3], [1,4], [2,3], [2,4]]
```

Internally, the `cross_operator` method is called.

Zip operator

The metaoperator `~Z` or `~Zop` zips two arrays and returns a new one.

```
[1,2] ~Z+ [3,4]    # [1+3, 2+4]
[1,2] ~Z  [3,4]    # [[1,3], [2,4]]
```

Internally, the `zip_operator` method is called.

Wise operator

Almost equivalent with the zip metaoperator, it does element-wise folding on two arbitrary nested arrays, where both arrays must have the same structure.

```
[1,2]       ~W  [3,4]         # [[1,3], [2,4]]
[1,2]       ~W+ [3,4]         # [1+3, 2+4]
[[[1]],[2]] ~W+ [[[3]],[4]]   # [[[1+3]], [2+4]]
```

Internally, the `wise_operator` method is called.

Scalar operator

The scalar operator applies a given operator to the elements of an arbitrary nested array, where the provided scalar is used as the second operand to the given operator.

```
[1,2,3]       ~S  5  # [[1,5], [2,5], [3,5]]
[1,2,3]       ~S* 5  # [1*5, 2*5, 3*5]
[1,[[2,[3]]]] ~S+ 5  # [1+5, [[2+5, [3+5]]]]
```

Internally, the `scalar_operator` method is called.

Reverse scalar operator

The reverse scalar operator uses the given scalar as a first operand to the given operator and is also defined for arbitrary nested arrays.

```
[3,4,5]       ~RS  1  # [[1,3], [1,4], [1,5]]
[3,4,5]       ~RS/ 1  # [1/3, 1/4, 1/5]
[3,[[4,[5]]]] ~RS/ 1  # [1/3, [[1/4, [1/5]]]]
```

Internally, the `rscalar_operator` method is called.

Variables

Variables are commonly declared using the `var` keyword:

```
var num = 42
var str = "42"
var bool = true
```

Lexical variables

These kinds of variables are lexical, but statically block scoped. This is the usual way of declaring variables in Sidef.

```
var x = 42    # sets the lexical x to 42
say x         # prints the lexical value of x
```

Static variables

This kind of variables are static, block-scoped and initialized only once.

```
static x = 42   # sets the static x to 42
say x           # prints the static value of x
```

Global variables

Global variables are declared at the top-level of the current namespace. They can be accessed from everywhere, anytime. However, try to avoid using them, unless you really don't have any better alternative.

```
global x = 42     # sets global x to 42
say x             # prints the global value of x
```

Local variables

Local variables (also known as "dynamically scoped variables") can be used to localize array/hash lvalues or global variables to a limited scope.

```
global x = 42    # sets the global x to 42
do {
   local x = 100     # localizes x inside this block to 100
   say x             # prints the local value of x
}
say x            # prints the global value of x (42)
```

A slightly more advanced example, illustrating the localization of an hash lvalue, would be:

```
func foo(h) {
    say h{:key}
}

var h = Hash(key => "a")

foo(h)                          # prints "a"
do {
    local h{:key} = "b"         # local change only
    foo(h)                      # prints: "b"
}
foo(h)                          # prints: "a"
```

Variable scoping

All variables (including functions and classes) are block scoped in the following way:

```
var x = 'o'

do {
    say x               # o
    var x = 'm'
    say x               # m
    do {
        say x           # m
        var x = 'b'
        say x           # b
    }
    say x               # m
}

say x                   # o
```

Declaring multiple variables at once is also possible:

```
var (x, y, z) = (3.14, false, "foo")
```

We can, also, declare variables with some default values:

```
var (x, y=755, z=777) = (666, 655)

say x        # prints: 666
say y        # prints: 655
say z        # prints: 777
```

Slurpy variables

Slurpy (or greedy) variables are a special type of variables which can be initialized with a list of values, creating automatically a container to hold the data.

```
var *arr = (1,2,3)    # creates an Array
say arr               # prints: [1,2,3]

var :hash = (a => 1, b => 2)   # creates an Hash
say hash                       # prints: Hash(a=>1, b=>2)
```

Working with variables

Any method applied to a variable is applied on the object at which the variable is pointing at:

```
var x = 'sidef'
say x.uc        # prints: `SIDEF`
say x           # prints: `sidef`
```

Special `!` at the end of a method changes the variable in-place (almost like in Ruby):

```
var x = 'sidef'
x.uc!            # notice the `!`
say x            # prints: `SIDEF`
```

Appending the = sign at the end of arithmetic operators, the variable will be changed in place:

```
var x = 5
x += 10            # adds 10 to "x"
say x              # prints: 15
```

The special operator `:=` (also available as `\\=`), assigns a value to a variable if the current value of the variable is `nil`:

```
var x = nil
x := 42           # assigns 42 to x if x is nil
x := 99           # x is already defined
say x             #=> 42
```

The defined-or operator `\\` can be used for checking if a variable is defined or not:

```
var x = nil         # nil represents an undefined value
say defined(x)      # prints 'false'

x \\ say 'undefined'        # prints 'undefined'
x \\= 99                    # sets x to 99
x \\ say 'undefined'        # no longer prints 'undefined'
```

Special identitiers

- `ARGV` is an Array that contains the program's command-line arguments, that were not given to Sidef.
- `ENV` is an Hash copy of environment variables and their values when the program was started.
- `ARGF` is a FileHandle object used to read lines from argument-files or from `STDIN` when no argument has been specified.
- `DATA` is a FileHandle object that points to the data stored after the `__END__` or `__DATA__` tokens.

```
say ARGV        # command-line arguments
say ENV{:HOME}  # get an environment variable

ARGF.each { |line|     # cat-like program
    say line
}

DATA.each {|line|       # iterate over lines in __DATA__
    say line
}

__DATA__
hello
world
```

Topic variable

The special topic variable (`_`) is declared at compile-time in each block-object in the program. You may not see its real name very often, because it has been overtaken by the elegant prefix dot (.) operator:

```
[25,36,49].map {.sqrt} \
          .each{.log.say}
```

...where `.sqrt` really means `_.sqrt`, and `.log.say` means `_.log.say`.

Variable references

Like in other programming languages, we have the capability of taking references to variables, by using the prefix backslash (`\`) operator for referencing and the prefix asterix (`*`) operator for dereferencing:

```
var name = "sidef"
var ref = \name
var original = *ref
```

Variable references are useful when passing them to functions (or methods) for assigning values.

```
func assign2ref (ref, value) {
    *ref = value
}

var x = 10
assign2ref(\x, 20)
say x                # prints: 20
```

There is also the possibility of taking references to the lvalues of an array or hash:

```
var a = [41, 42, 43]
var r = \a[1]           # reference of the second element

*r = 99                 # changes the value inside the array
say a                   # prints: [41, 99, 43]
```

Constants

Sidef implements three kinds of constants: `const`, `define` and `enum`.

const

The common way of declaring constants in Sidef, is by using the `const` keyword:

```
const pi = 3.14
say pi                  # prints: 3.14
#pi = 3                 # compile-time error: can't modify non-lvalue constant
```

This kind of constants are created dynamically at run-time, but cannot be changed during the execution of the program.

When declared inside a class or a function, the constant is created and initialized dynamically, as illustrated in the following example:

```
func f(a) {
    const x = a          # created dynamically at each function call
    return (x + 2)
}

say f(40)       #=> 42
say f(50)       #=> 52
```

define

This keyword will define a compile-time evaluated constant and will point directly to the object at which it evaluated to.

```
define PHI =   (1.25.sqrt + 0.5)
define IHP = (-(1.25.sqrt - 0.5))

say (PHI**7 - IHP**7 / PHI-IHP)
```

This type of constants are the most efficient ones.

enum

The `enum` keyword will automatically declare and assign a list of constants with ascending numeric values (starting from 0):

```
enum |Black, White|
say Black             # prints: 0
say White             # prints: 1
```

Alternatively, we have the possibility for specifying an initial value, which will get incremented after each declaration, by calling the method `.inc()`.

```
enum |α="a", β|
say α             # prints: 'a'
say β             # prints: 'b'
```

Multidimensional arrays

Multidimensional arrays can be defined as:

```
var A = [
          [1, 2],
          [3, 4],
          [5, 6],
          [7, 8],
        ]

var B = [
          [1, 2, 3],
          [4, 5, 6],
        ]
```

In Sidef, we have the `Array.wise_op()` method, which takes two arbitrary nested arrays and an operator, folding each element (entrywise) with the provided operator, which is also available as `a ~Wop b`:

```
say ([1,2,[3,[4]]] ~W+ [42,43,[44,[45]]])       #=> [43, 45, [47, [49]]]
```

Alternatively:

```
say wise_op([1,2,[3,[4]]], '+', [42,43,[44,[45]]])   #=> [43, 45, [47, [49]]]
```

Scalar operations:

```
A `scalar_add` 42   # scalar addition       (aliased as `sadd`)
A `scalar_sub` 42   # scalar subtraction    (aliased as `ssub`)
A `scalar_mul` 42   # scalar multiplication (aliased as `smul`)
A `scalar_div` 42   # scalar division       (aliased as `sdiv`)
```

This methods are provided by `Array.scalar_op()`, which, just like `Array.wise_op()`, also supports arbitrary nested arrays:

```
say ([1,2,[3,[4]]] ~S+ 42)   #=> [43, 44, [45,  [46]]]
say ([1,2,[3,[4]]] ~S* 42)   #=> [42, 84, [126, [168]]]
```

...which is equivalent with:

```
say scalar_op([1,2,[3,[4]]], '+', 42)  #=> [43, 44, [45,  [46]]]
say scalar_op([1,2,[3,[4]]], '*', 42)  #=> [42, 84, [126, [168]]]
```

Iteration over 2D arrays

The extended `for-in` loop provides support for iterating over a 2D-array, which is useful in combination with the cross and zip metaoperators:

```
for a,b in ([1,2] ~X [3,4]) {
    say "#{a} #{b}"
}
```

This is equivalent with:

```
[[1,2], [3,4]].cartesian {|a,b|
    say "#{a} #{b}"
}
```

and outputs:

```
1 3
1 4
2 3
2 4
```

Blocks

In Sidef, a block of code is an object which encapsulates zero or more expressions and is delimited by a pair of curly braces (`{}`).

```
var block = {
    say "Hello, World!"
}
```

Block parameters

For declaring block parameters, Sidef borrows Ruby's way of doing this, by using the `|arg1, arg2, ...|` special syntax:

```
{ |a, b|

   say a            # prints: 1
   say b            # prints: 2

}(1, 2)
```

Callbacks

Blocks are also used as arguments to many built-in methods as callback blocks:

```
{ print "Sidef! " } * 3        # prints "Sidef! Sidef! Sidef! "
5.times {|x| print x }         # prints "01234"
[1,2,3].sort {|a,b| b <=> a }  # returns a new array: [3,2,1]
```

The `Block` class also implements some useful methods, such as:

```
say {|n| n**2 }.map(1..5)        #=> [1, 4, 9, 16, 25]
say { .is_odd }.grep([1,2,3,4])  #=> [1, 3]
```

Functions

Like mathematical functions, Sidef's functions can be recursive, take arguments and return values.

```
func hello (name) {
   say "Hello, #{name}!"
}

hello("Sidef")
```

The special `__FUNC__` keyword refers to the current function:

```
func factorial (n) {
   if (n > 1) {
       return (n * __FUNC__(n - 1))
   }
   return 1
}

say factorial(5)     # prints: 120
```

Closures

In Sidef, all functions are first-class objects which can be passed around like any other object. Additionally, all functions (including methods) are lexical closures.

```
func curry(f, *args1) {
    func (*args2) {
        f(args1..., args2...)
    }
}

func add(a, b) {
    a + b
}

var adder = curry(add, 1)
say adder(3)                #=> 4
```

Caching functions

By specifying the `cached` trait in a function (or method) declaration, Sidef will automatically cache it for you.

```
func fib(n) is cached {
    return n if (n <= 1)
    fib(n-1) + fib(n-2)
}
say fib(100)              #=> 354224848179261915075
```

Optional arguments

The parameters of a function can also be declared with a default value, which provides support for optional arguments.

```
func hello (name="Sidef") {
    say "Hello, #{name}!"
}

hello()           # prints: "Hello, Sidef!"
hello("World")    # prints: "Hello, World!"
```

The default value of a parameter is evaluated only when an argument is not provided for that particular parameter, and it can be any arbitrary expression:

```
func foo (a = 1.25.sqrt, b = 1/2) {
     a + b
}

say foo()           # prints the result of: sqrt(1.25) + 1/2
say foo(21, 21)     # prints: 42
```

Named parameters

This is a very nice feature borrowed from Perl 6 which allows a function to be called with named parameters, giving us the flexibility to put the arguments in no specific order:

```
func divide(a, b) {
   a / b
}

say divide(b: 5, a: 35)  # prints: 7
```

Variadic functions

A slurpy variable in the form of `*name` can be used as a function parameter to collect the remaining arguments inside an array:

```
func f(*args) {
   say args       #=> [1, 2, 3]
}

f(1, 2, 3)
```

Alternatively, by using a named variable in the form of `:name`, the arguments are collected inside an hash:

```
func f(:pairs) {
   say pairs      #=> Hash(a=>1, b=>2)
}

f(a => 1, b => 2)
```

Typed parameters

A function can be declared with typed parameters, which are checked at run-time.

```
func concat(String a, String b) {
    a + b
}
```

Now, the function can only be called with strings as arguments:

```
say concat("o", "k")  # ok
say concat(1, 2)      # runtime error
```

The typed parameters require a specific type of object, but they do not default to anything when no value is provided. This means that all typed-parameters are mandatory, unless a default value is provided:

```
func concat(String a="foo", String b="bar") {
   a + b
}

say concat()          # prints: "foobar"
say concat("mini")    # prints: "minibar"
say concat(1, 2)      # this is still a run-time error
```

Modules

In Sidef, a module is the declaration of a new namespace:

```
module Fibonacci {
    func nth(n) {
        n > 1 ? nth(n-2)+nth(n-1) : n
    }
}

say Fibonacci::nth(12)    # prints: 144
```

The default namespace name is `main`. Variables from other namespaces can be used inside a module by either importing them, or by specifying their full name which includes the namespace:

```
var foo = 42

module Bar {
    var baz = 99
    say main::foo   #=> 42
}

say Bar::baz        #=> 99
```

Importing an identifier in the current namespace, can be done using the syntax `import namespace::identifier_name`:

```
var foo = 42

module Bar {
    import main::foo
    var baz = 2*foo
}

import Bar::baz
say baz             #=> 84
```

Modules cannot be instantiated as objects, cannot be modified, and cannot be directly referred to by their name (like `ModuleName`) without a member name (like `ModuleName::var_name`). Therefore, they are for containing related code that doesn't need to be instantiated like a class.

Classes

A class is a declaration of a constructor of objects with a specific type. Each object has zero or more attributes (instance variables) with zero or more behaviours (methods), that are defined inside a specific class or inherited from super-classes.

```
class Person (name, age, address) {
    method position {
        # GPS.locate(self.address)
    }

    method increment_age(amount=1) {
         self.age += amount
    }
}

var obj = Person(
       name: "Foo",
        age: 50,
    address: "St. Bar"
)

say obj.age                # prints: 50
say obj.name               # prints: "Foo"
say obj.address            # prints: "St. Bar"

obj.name = "Baz"           # changes name to "Baz"
say obj.name               # prints: "Baz"

obj.increment_age          # increments age by 1
say obj.age                # prints: 51
```

Class attributes

The attributes of a class can be either specified as parameters, or declared with the `has` keyword.

```
class Example(a, b) {
   has c = 3
   has d = a+c
}

var obj = Example(1, 2)

say obj.a   #=> 1
say obj.b   #=> 2
say obj.c   #=> 3
say obj.d   #=> 4
```

Class initialization

Extra object-initialization setup can be done by defining a method named `init`, which will be called automatically called whenever a new instance-object is created.

```
class Example (a, b) {

   has r = 0

   method init {      # called automatically
      r = a+b
   }

   method foo {
      r
   }
}

var obj = Example(3, 4)
say obj.foo              #=> 7
```

Class variables

The syntax `ClassName!var_name` can be used for defining, accessing or modifying a class variable.

```
class Example {

    Example!hidden = 'secret'    # global class variable

    method concat (str) {
        str + ' ' + Example!hidden
    }
}

var x = Example()
var y = Example()

say x.concat('foo')   #=> 'foo secret'
say y.concat('bar')   #=> 'bar secret'

Example!hidden = 'public'  # changing the class variable

say x.concat('foo')   #=> 'foo public'
say y.concat('bar')   #=> 'bar public'
```

The modification of a class variable can be localized by prefixing the declaration with the `local` keyword:

```
local Example!hidden = 'local value'
```

Class inheritance (experimental)

Inheritance of behaviors and attributes, by a given class, is declared with the `<` operator, followed by the name of the class from which the current class inherits:

```
class Animal(String name, Number age)  {
    method speak { "..." }
}

class Dog(String color) < Animal {
    method speak { "woof" }
    method ageHumanYears { self.age * 7 }
}

class Cat < Animal {
    method speak { "meow" }
}

var dog = Dog(name: "Sparky", age: 6, color: "white")
var cat = Cat(name: "Mitten", age: 3)

say dog.speak          #=> woof
say cat.speak          #=> meow
say cat.age            #=> 3
say dog.ageHumanYears  #=> 42
say dog.color          #=> white
```

Multiple inheritance is declared with the `<<` operator, followed by two or more class names, separated by commas:

```
class Camera { }
class MobilePhone { }
class CameraPhone << Camera, MobilePhone { }
```

Types

In Sidef, a new type can be declared as a class and its name can be used in function or method parameters to provide dynamic type checking:

```
class Point(Number x, Number y) {
    method to_s { "Point(#{x}, #{y})" }      # auto-stringification
}

func foo(Point p) {
    say p
}

foo(Point(5, 6))
```

A type can be veryfied by using the `.kind_of(TYPE)` method, which is defined inside the `Object` class, which is the parent of all classes (including user-defined classes):

```
say "foo".kind_of(String)     #=> true
say "bar".kind_of(Number)     #=> false
```

The type of an object can be found usind the following two methods (which are also defined inside the `Object` class):

```
say "foo".ref           #=> Sidef::Types::String::String
say "foo".class         #=> String
```

Subsets

A subset is a definition which specifies the upper limit of inheritance, with optional argument validation.

```
subset Integer     < Number  { |n| n.is_int }
subset Natural     < Integer { |n| n.is_pos }
subset EvenNatural < Natural { |n| n.is_even }

func foo(n < EvenNatural) {
     say n
}

foo(42)         # ok
foo(43)         # failed assertion at run-time
```

In some sense, a subset is the opposite of a type. For example, let's consider the following class hierarchy:

```
class Hello(name) {
    method greet { say "Hello, #{self.name}!" }
}

class Hi < Hello {
    method greet { say "Hi, #{self.name}!" }
}

class Hey < Hi {
    method greet { say "Hey, #{self.name}!" }
}
```

If we declare a function that accepts a subset of `Hi`, it will accept `Hello`, but it cannot accept `Hey`:

```
func greet(obj < Hi) { obj.greet }       # `Hi` is the upper limit

greet(Hi("Foo"))        # ok
greet(Hello("Bar"))     # ok
greet(Hey("Baz"))       # fail: `Hey` is too evolved
```

On the other hand, if we use `Hi` as a type assertion, it will accept `Hey`, but not `Hello`:

```
func greet(Hi obj) { obj.greet }         # `Hi` is the lower limit

greet(Hi("Foo"))        # ok
greet(Hey("Baz"))       # ok
greet(Hello("Bar"))     # fail: `Hello` is too primitive
```

Subsets can also be used for combining multiple types into one type, creating an union type:

```
subset StrNum < String, Number

func concat(a < StrNum, b < StrNum) {
    a + b
}

say concat("o", "k")       # ok
say concat(13, 29)         # 42
say concat([41], [42])     # runtime error
```

Lazy evaluation

Lazy evaluation is a common feature in functional programming languages and provides an way to delay the evaluation of an expression until the result is actually needed.

By default, Sidef is an eagerly evaluated language, but it supports a form of lazy evaluation, provided by the method `Object.lazy()`, almost in the same way as Ruby does:

```
say (^Inf -> lazy.grep{ .is_prime }.first(10))     # first 10 primes
```

The `.lazy` method returns a Lazy object, which behaves almost like an Array, except that it executes the methods in a pipeline fashion and dynamically stops when no more elements are required, without creating any temporary arrays in memory:

```
for line in (DATA.lazy.grep{ _ < 'd' }.map{ print ">> "; .uc }) {
   say line
}

__DATA__
a
b
c
d
```

Output (which illustrates that `.map{}` is truly lazy):

```
>> A
>> B
>> C
```

This mechanism is generic enough to support any kind of object that implements the `.iter()` method, which returns a Block that gives back one element at a time when it's called with no arguments. When the iteration ends, it must return `nil`.

```
class Example(data) {
    method iter {
        var p = 0
        {
            data[p++]
        }
    }
}

var obj = Example([1,2,3,4,5])
say obj.lazy.grep{.is_prime}.to_a       # filters all the primes lazily
```

Currently, the `.iter()` method is defined in the following built-in classes: Array, String, FileHandle, DirHandle, RangeString, RangeNumber and Lazy.

Lazy methods

Lazy methods provide an interesting concept of partially applying a method on a given object, delaying the evaluation until the result is needed.

```
var lz = 42.method('>')        #=> LazyMethod
say lz(41)                     #=> true (i.e.: 42 > 41)
```

`Object.method()` returns a `LazyMethod` object which can be called just like any other function, producing the result by calling the method which is stored inside the `LazyMethod` object.

This allow us to store or pass around partial expressions, which can be evaluated multiple times at any point in the future:

```
var str = "a-b-c"
var lzsplit = str.method(:uc).method(:split)

say lzsplit('')         #=> ["A", "-", "B", "-", "C"]
say lzsplit('-')        #=> ["A", "B", "C"]
```

Multiple dispatch

Multiple dispatch allows us to declare multiple variants of the same function or method, each working only with a certain type of arguments.

For example, if we declare two functions with the same name, but with different types of parameters, Sidef will decide automatically which function to call:

```
func foo(Number a) {
    say a
}

func foo(String a) {
    say a
}

foo(1234)       # calls the first function
foo("bar")      # calls the second function
```

The functions are checked in the order in which they are declared:

```
func foo(Array a) { ... }       # works with an array
func foo(Hash h)  { ... }       # works with an hash
func foo(any)     { ... }       # works with anything else
```

It also works with methods:

```
class Example {
    method foo(Number n, String s) {
        say "first"
    }

    method foo(Array a) {
        say "second"
    }
}

var obj = Example()
obj.foo(1234, "foo")    # calls the first method
obj.foo([1,2,3])        # calls the second method
```

Functional pattern matching

The functional pattern matching it's an extension of the multiple dispatch feature, and allows us to call a certain function or method using values or expressions as patterns, instead of types.

Example:

```
func foo((1)) { say "one" }
func foo((2)) { say "two" }
func foo (n)  { say n }

foo(1)      # calls the first function
foo(2)      # calls the second function
foo(3)      # calls the third function
```

Taking advantage of this feature, we can write the Fibonacci function in the following way:

```
func fib((0)) { 0 }
func fib((1)) { 1 }
func fib (n)  { fib(n-1) + fib(n-2) }

say fib(12)    # prints: 144
```

In addition, instead of an expression, we can specify a block of code, which will get called with the value of the argument for checking:

```
func fib({.is_neg})  { NaN }
func fib({.is_zero}) { 0 }
func fib({.is_one})  { 1 }
func fib(n)          { fib(n-1) + fib(n-2) }

say fib(12)    # prints: 144
```

For keeping the value of the argument, we can specify a parameter name in front of the block used for pattern matching:

```
func fib(n { _ <= 1 }) { n }
func fib(n)            { fib(n-1) + fib(n-2) }

say fib(12)    # prints: 144
```

Pattern matching is available for methods as well:

```
class Ackermann {
    method A({ .is_zero }, n) {
        n + 1
    }

    method A(m, (0)) {
        self.A(m-1, 1)
    }

    method A(m, n) {
        self.A(m-1, self.A(m, n-1))
    }
}

var obj = Ackermann()
say obj.A(3, 2)             # prints: 29
```

Programming tasks

This is an automatically generated section. It contains programming tasks from Rosettacode implemented in Sidef.

The scripts used for creating this section of the book, can be found by clicking on the following link:

- https://github.com/trizen/perl-scripts/tree/master/Book tools

Enjoy!

10001th prime

```
say 10001.prime
```

Output:

```
104743
```

100 doors

Unoptimized

```
var doors = []

{ |pass|
    { |i|
        if (pass `divides` i) {
            doors[i] := false -> not!
        }
    } << 1..100
} << 1..100

{ |i|
    say ("Door %3d is %s" % (i, doors[i] ? 'open' : 'closed'))
} << 1..100
```

Optimized

```
{ |i|
    "Door %3d is %s\n".printf(i, <closed open>[i.is_sqr])
} << 1..100
```

24 game

```
const digits  = (1..9 -> pick(4))
const grammar = Regex(
    '^ (?&exp) \z
      (?(DEFINE)
          (?<exp> ( (?&term) (?&op) (?&term) )+ )
          (?<term> \( (?&exp) \) | [' + digits.join + '])
          (?<op> [\-+*/] )
      )', 'x'
)

say "Here are your digits: #{digits.join(' ')}"

loop {
    var input = read("Expression: ", String)

    var expr = input
    expr -= /\s+/g     # remove all whitespace

    if (input == 'q') {
        say "Goodbye.  Sorry you couldn't win."
        break
    }

    var given_digits = digits.map{.to_s}.sort.join
    var entry_digits = input.scan(/\d/).sort.join

    if ((given_digits != entry_digits) || (expr !~ grammar)) {
        say "That's not valid"
        next
    }

    given(var n = eval(input)) {
        when (24) { say "You win!"; break }
        default   { say "Sorry, your expression is #{n}, not 24" }
    }
}
```

Output:

```
Here are your digits: 8 2 3 4
Expression: 8 * (2 - (3 + 4))
Sorry, your expression is -40, not 24
Expression: 8 * (2 - (3 -
That's not valid
Expression: 8 * (2 - (3 - 4))
You win!
```

24 game/Solve

With evaluation:

```
var formats = [
    '((%d %s %d) %s %d) %s %d',
    '(%d %s (%d %s %d)) %s %d',
    '(%d %s %d) %s (%d %s %d)',
    '%d %s ((%d %s %d) %s %d)',
    '%d %s (%d %s (%d %s %d))',
]

var op = %w( + - * / )
var operators = op.map { |a| op.map {|b| op.map {|c| "#{a} #{b} #{c}" } } }.flat

loop {
    var input = read("Enter four integers or 'q' to exit: ", String)
    input == 'q' && break

    if (input !~ /^\h*[1-9]\h+[1-9]\h+[1-9]\h+[1-9]\h*$/) {
        say "Invalid input!"
        next
    }

    var n = input.split.map{.to_n}
    var numbers = n.permutations

    formats.each { |format|
        numbers.each { |n|
            operators.each { |operator|
                var o = operator.split;
                var str = (format % (n[0],o[0],n[1],o[1],n[2],o[2],n[3]))
                eval(str) == 24 && say str
            }
        }
    }
}
```

Without evaluation:

```
var formats = [
    {|a,b,c|
        Hash(
            func   => {|d,e,f,g| ((d.$a(e)).$b(f)).$c(g) },
            format => "((%d #{a} %d) #{b} %d) #{c} %d"
        )
    },
    {|a,b,c|
        Hash(
            func   => {|d,e,f,g| (d.$a((e.$b(f)))).$c(g) },
            format => "(%d #{a} (%d #{b} %d)) #{c} %d",
        )
    },
    {|a,b,c|
        Hash(
            func   => {|d,e,f,g| (d.$a(e)).$b(f.$c(g)) },
            format => "(%d #{a} %d) #{b} (%d #{c} %d)",
        )
    },
    {|a,b,c|
        Hash(
            func   => {|d,e,f,g| (d.$a(e)).$b(f.$c(g)) },
            format => "(%d #{a} %d) #{b} (%d #{c} %d)",
        )
    },
    {|a,b,c|
        Hash(
            func   => {|d,e,f,g| d.$a(e.$b(f.$c(g))) },
            format => "%d #{a} (%d #{b} (%d #{c} %d))",
        )
    },
];

var op = %w( + - * / )
var blocks = op.map { |a| op.map { |b| op.map { |c| formats.map { |format|
    format(a,b,c)
}}}}.flat

loop {
    var input = Sys.scanln("Enter four integers or 'q' to exit: ");
    input == 'q' && break;

    if (input !~ /^\h*[1-9]\h+[1-9]\h+[1-9]\h+[1-9]\h*$/) {
        say "Invalid input!"
        next
    }

    var n = input.split.map{.to_n}
    var numbers = n.permutations

    blocks.each { |block|
        numbers.each { |n|
            if (block{:func}.call(n...) == 24) {
                say (block{:format} % (n...))
            }
        }
    }
}
```

Output:

```
Enter four integers or 'q' to exit: 8 7 9 6
(8 / (9 - 7)) * 6
(6 / (9 - 7)) * 8
(8 * 6) / (9 - 7)
(6 * 8) / (9 - 7)
8 / ((9 - 7) / 6)
6 / ((9 - 7) / 8)
8 * (6 / (9 - 7))
6 * (8 / (9 - 7))
Enter four integers or 'q' to exit: q
```

4-rings or 4-squares puzzle

```
func four_squares (list, unique=true, show=true) {

    var solutions = []

    func check(c) {
        solutions << c if ([
            c[0] + c[1],
            c[1] + c[2] + c[3],
            c[3] + c[4] + c[5],
            c[5] + c[6],
        ].uniq.len == 1)
    }

    if (unique) {
        list.combinations(7, {|*a|
            a.permutations { |*c|
                check(c)
            }
        })
    } else {
        7.of { list }.cartesian {|*c|
            check(c)
        }
    }

    say (solutions.len,
        (unique ? ' ' : ' non-'),
        "unique solutions found using #{list.join(', ')}.\n")

    if (show) {
        var f = "%#{list.max.len+1}s"
        say ("\n".join(
                ('a'..'g').map{f % _}.join,
                solutions.map{ .map{f % _}.join }...
        ), "\n")
    }
}

# TASK
four_squares(@(1..7))
four_squares(@(3..9))
four_squares([8, 9, 11, 12, 17, 18, 20, 21])
four_squares(@(0..9), unique: false, show: false)
```

Output:

```
8 unique solutions found using 1, 2, 3, 4, 5, 6, 7.

 a b c d e f g
 3 7 2 1 5 4 6
 4 5 3 1 6 2 7
 4 7 1 3 2 6 5
 5 6 2 3 1 7 4
 6 4 1 5 2 3 7
 6 4 5 1 2 7 3
 7 2 6 1 3 5 4
 7 3 2 5 1 4 6

4 unique solutions found using 3, 4, 5, 6, 7, 8, 9.

 a b c d e f g
 7 8 3 4 5 6 9
 8 7 3 5 4 6 9
 9 6 4 5 3 7 8
 9 6 5 4 3 8 7

8 unique solutions found using 8, 9, 11, 12, 17, 18, 20, 21.

  a  b  c  d  e  f  g
 17 21  8  9 11 18 20
 20 18 11  9  8 21 17
 17 21  9  8 12 18 20
 20 18  8 12  9 17 21
 20 18 12  8  9 21 17
 21 17  9 12  8 18 20
 20 18 11  9 12 17 21
 21 17 12  9 11 18 20

2860 non-unique solutions found using 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
```

99 bottles of beer

```
for i in (100 ^.. 0) {
    var bottles = "#{i == 0 ? 'No' : i} bottle#{i == 1 ? '' : 's'}"
    var sentence = "#{bottles} of beer on the wall" -> say
    if (i > 0) {
        say sentence.substr(0, bottles.length + 8)
        say "Take one down, pass it around\n"
    }
}
```

Simpler:

```
for n in (100 ^.. 2) {
  say "#{n} bottles of beer on the wall, #{n} bottles of beer!"
  say "Take one down, pass it around, #{n - 1} bottle#{n > 2 ? 's' : ''} of beer on the wall.\n"
}

say "One bottle of beer on the wall, one bottle of beer!"
say "Take one down, pass it around, no more bottles of beer on the wall."
```

9 billion names of God the integer

```
var cache = [[1]]

func cumu (n) {
    for l (cache.len .. n) {
        var r = [0]
        for i (1..l) {
            r << (r[-1] + cache[l-i][min(i, l-i)])
        }
        cache << r
    }
    cache[n]
}

func row (n) {
    var r = cumu(n)
    n.of {|i| r[i+1] - r[i] }
}

say "rows:"
for i (1..15) {
    "%2s: %s\n".printf(i, row(i))
}

say "\nsums:"

for i in [23, 123, 1234, 12345] {
    "%2s : %4s\n".printf(i, cumu(i)[-1])
}
```

Output:

```
rows:
 1: [1]
 2: [1, 1]
 3: [1, 1, 1]
 4: [1, 2, 1, 1]
 5: [1, 2, 2, 1, 1]
 6: [1, 3, 3, 2, 1, 1]
 7: [1, 3, 4, 3, 2, 1, 1]
 8: [1, 4, 5, 5, 3, 2, 1, 1]
 9: [1, 4, 7, 6, 5, 3, 2, 1, 1]
10: [1, 5, 8, 9, 7, 5, 3, 2, 1, 1]
11: [1, 5, 10, 11, 10, 7, 5, 3, 2, 1, 1]
12: [1, 6, 12, 15, 13, 11, 7, 5, 3, 2, 1, 1]
13: [1, 6, 14, 18, 18, 14, 11, 7, 5, 3, 2, 1, 1]
14: [1, 7, 16, 23, 23, 20, 15, 11, 7, 5, 3, 2, 1, 1]
15: [1, 7, 19, 27, 30, 26, 21, 15, 11, 7, 5, 3, 2, 1, 1]

sums:
23 : 1255
123 : 2552338241
1234 : 1569787972237332287878657223549599 30
^C
```

A* search algorithm

```
class AStarGraph {

    has barriers = [
```

```
    has barriers = [
        [2,4],[2,5],[2,6],[3,6],[4,6],[5,6],[5,5],[5,4],[5,3],[5,2],[4,2],[3,2]
    ]

    method heuristic(start, goal) {
        var (D1 = 1, D2 = 1)
        var dx = abs(start[0] - goal[0])
        var dy = abs(start[1] - goal[1])
        (D1 * (dx + dy)) + ((D2 - 2*D1) * Math.min(dx, dy))
    }

    method get_vertex_neighbours(pos) {
        gather {
            for dx, dy in [[1,0],[-1,0],[0,1],[0,-1],[1,1],[-1,1],[1,-1],[-1,-1]] {
                var x2 = (pos[0] + dx)
                var y2 = (pos[1] + dy)
                (x2<0 || x2>7 || y2<0 || y2>7) && next
                take([x2, y2])
            }
        }
    }

    method move_cost(_a, b) {
        barriers.contains(b) ? 100 : 1
    }
}

func AStarSearch(start, end, graph) {

    var G = Hash()
    var F = Hash()

    G{start} = 0
    F{start} = graph.heuristic(start, end)

    var closedVertices = []
    var openVertices = [start]
    var cameFrom = Hash()

    while (openVertices) {

        var current = nil
        var currentFscore = Inf

        for pos in openVertices {
            if (F{pos} < currentFscore) {
                currentFscore = F{pos}
                current = pos
            }
        }

        if (current == end) {
            var path = [current]
            while (cameFrom.contains(current)) {
                current = cameFrom{current}
                path << current
            }
            path.flip!
            return (path, F{end})
        }

        openVertices.remove(current)
        closedVertices.append(current)

        for neighbour in (graph.get_vertex_neighbours(current)) {
            if (closedVertices.contains(neighbour)) {
```

```
                next
            }
            var candidateG = (G{current} + graph.move_cost(current, neighbour))

            if (!openVertices.contains(neighbour)) {
                openVertices.append(neighbour)
            }
            elsif (candidateG >= G{neighbour}) {
                next
            }

            cameFrom{neighbour} = current
            G{neighbour} = candidateG
            var H = graph.heuristic(neighbour, end)
            F{neighbour} = (G{neighbour} + H)
        }
    }

    die "A* failed to find a solution"
}

var graph = AStarGraph()
var (route, cost) = AStarSearch([0,0], [7,7], graph)

var w = 10
var h = 10

var grid = h.of { w.of { "." } }
for y in (^h) { grid[y][0] = "█"; grid[y][-1] = "█" }
for x in (^w) { grid[0][x] = "█"; grid[-1][x] = "█" }

for x,y in (graph.barriers) { grid[x+1][y+1] = "█" }
for x,y in (route)          { grid[x+1][y+1] = "x" }

grid.each { .join.say }

say "Path cost #{cost}: #{route}"
```

Output:

```
██████████
█x.......█
█.x......█
█..x.███.█
█.x█...█.█
█.x█...█.█
█.x█████.█
█..xxxxx.█
█.......x█
██████████
Path cost 11: [[0, 0], [1, 1], [2, 2], [3, 1], [4, 1], [5, 1], [6, 2], [6, 3], [6, 4], [6, 5], [6, 6],
```

A+B

Works with both positive and negative integers.

```
say STDIN.readline.words.map{.to_i}.sum
```

More idiomatically:

```
say read(String).words»to_i()»«+»
```

Explicit summation:

```
var (a, b) = read(String).words.map{.to_i}...
say a+b
```

ABC problem

```
func can_make_word(word, blocks) {

    blocks.map! { |b| b.uc.chars.sort.join }.freq!

    func(word, blocks) {
        var char = word.shift
        var candidates = blocks.keys.grep { |k| 0 <= k.index(char) }

        for candidate in candidates {
            blocks{candidate} <= 0 && next;
            local blocks{candidate} = (blocks{candidate} - 1);
            return true if (word.is_empty || __FUNC__(word, blocks));
        }

        return false;
    }(word.uc.chars, blocks)
}
```

Tests:

```
var b1 = %w(BO XK DQ CP NA GT RE TG QD FS JW HU VI AN OB ER FS LY PC ZM)
var b2 = %w(US TZ AO QA)

var tests = [
    ["A", true, b1],
    ["BARK", true, b1],
    ["BOOK", false, b1],
    ["TREAT", true, b1],
    ["COMMON", false, b1],
    ["SQUAD", true, b1],
    ["CONFUSE", true, b1],
    ["auto", true, b2],
];

tests.each { |t|
    var bool = can_make_word(t[0], t[2]);
    say ("%7s -> %s" % (t[0], bool));
    assert(bool == t[1])
}
```

Output:

```
      A -> true
   BARK -> true
   BOOK -> false
  TREAT -> true
 COMMON -> false
  SQUAD -> true
CONFUSE -> true
   auto -> true
```

Abstract type

```
class A {
    # must be filled in by the class which will inherit it
    method abstract() { die 'Unimplemented' };

    # can be overridden in the class, but that's not mandatory
    method concrete() { say '# 42' };
}

class SomeClass << A {
    method abstract() {
        say "# made concrete in class"
    }
}

var obj = SomeClass.new;
obj.abstract();   # made concrete in class
obj.concrete();   # 42
```

Abundant, deficient and perfect number classifications

```
func propdivsum(n) {
    n.sigma - n
}

var h = Hash()
{|i| ++(h{propdivsum(i) <=> i} := 0) } << 1..20000
say "Perfect: #{h{0}}    Deficient: #{h{-1}}    Abundant: #{h{1}}"
```

Output:

```
Perfect: 4    Deficient: 15043    Abundant: 4953
```

Abundant odd numbers

```
func is_abundant(n) {
    n.sigma > 2*n
}

func odd_abundants (from = 1) {
     from =  (from + 2)//3
     from += (from%2 - 1)
     3*from .. Inf `by` 6 -> lazy.grep(is_abundant)
}

say         " Index |      Number | proper divisor sum"
const sep = "-------+-------------+-------------------\n"
const fstr = "%6s | %11s | %11s\n"

print sep

odd_abundants().first(25).each_kv {|k,n|
    printf(fstr, k+1, n, n.sigma-n)
}

with (odd_abundants().nth(1000)) {|n|
    printf(sep + fstr, 1000, n, n.sigma-n)
}

with(odd_abundants(1e9).first) {|n|
    printf(sep + fstr, '***', n, n.sigma-n)
}
```

Output:

```
 Index |      Number | proper divisor sum
-------+-------------+-------------------
     1 |         945 |         975
     2 |        1575 |        1649
     3 |        2205 |        2241
     4 |        2835 |        2973
     5 |        3465 |        4023
     6 |        4095 |        4641
     7 |        4725 |        5195
     8 |        5355 |        5877
     9 |        5775 |        6129
    10 |        5985 |        6495
    11 |        6435 |        6669
    12 |        6615 |        7065
    13 |        6825 |        7063
    14 |        7245 |        7731
    15 |        7425 |        7455
    16 |        7875 |        8349
    17 |        8085 |        8331
    18 |        8415 |        8433
    19 |        8505 |        8967
    20 |        8925 |        8931
    21 |        9135 |        9585
    22 |        9555 |        9597
    23 |        9765 |       10203
    24 |       10395 |       12645
    25 |       11025 |       11946
-------+-------------+-------------------
  1000 |      492975 |      519361
-------+-------------+-------------------
   *** |  1000000575 |  1083561009
```

Accumulator factory

```
class Accumulator(sum) {
    method add(num) {
        sum += num
    }
}

var x = Accumulator(1)
x.add(5)
Accumulator(3)
say x.add(2.3)              # prints: 8.3
```

The same thing can be achieved by returning a closure from the *Accumulator* function.

```
func Accumulator(sum) {
    func(num) { sum += num }
}

var x = Accumulator(1)
x(5)
Accumulator(3)
say x(2.3)                  # prints: 8.3
```

Ackermann function

```
func A(m, n) {
    m == 0 ? (n + 1)
           : (n == 0 ? (A(m - 1, 1))
                     : (A(m - 1, A(m, n - 1))));
}
```

Alternatively, using multiple dispatch:

```
func A((0), n) { n + 1 }
func A(m, (0)) { A(m - 1, 1) }
func A(m,  n)  { A(m-1, A(m, n-1)) }
```

Calling the function:

```
say A(3, 2);     # prints: 29
```

Add a variable to a class instance at runtime

```
class Empty{};
var e = Empty();    # create a new class instance
e{:foo} = 42;       # add variable 'foo'
say e{:foo};        # print the value of 'foo'
```

Additive primes

```
func additive_primes(upto, base = 10) {
    upto.primes.grep { .sumdigits(base).is_prime }
}

additive_primes(500).each_slice(10, {|*a|
    a.map { '%3s' % _ }.join(' ').say
})
```

Output:

```
  2   3   5   7  11  23  29  41  43  47
 61  67  83  89 101 113 131 137 139 151
157 173 179 191 193 197 199 223 227 229
241 263 269 281 283 311 313 317 331 337
353 359 373 379 397 401 409 421 443 449
461 463 467 487
```

Address of a variable

```
var n = 42;
say Sys.refaddr(\n);        # prints the address of the variable
say Sys.refaddr(n);         # prints the address of the object at which the variable points to
```

Output:

```
42823224
37867184
```

AKS test for primes

```
func binprime(p) {
    p >= 2 || return false
    for i in (1 .. p>>1) {
        (binomial(p, i) % p) && return false
    }
    return true
}

func coef(n, e) {
    (e == 0) && return "#{n}"
    (n == 1) && (n = "")
    (e == 1) ? "#{n}x" : "#{n}x^#{e}"
}

func binpoly(p) {
    join(" ", coef(1, p), ^p -> map {|i|
        join(" ", %w(+ -)[(p-i)&1], coef(binomial(p, i), i))
    }.reverse...)
}

say "expansions of (x-1)^p:"
for i in ^10 { say binpoly(i) }
say "Primes to 80: [#{2..80 -> grep { binprime(_) }.join(' ')}]"
```

Output:

```
expansions of (x-1)^p:
1
x - 1
x^2 - 2x + 1
x^3 - 3x^2 + 3x - 1
x^4 - 4x^3 + 6x^2 - 4x + 1
x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1
x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1
x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1
x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1
x^9 - 9x^8 + 36x^7 - 84x^6 + 126x^5 - 126x^4 + 84x^3 - 36x^2 + 9x - 1
Primes to 80: [2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79]
```

Align columns

```
class Format(text, width) {
    method align(j) {
        text.map { |row|
            row.range.map { |i|
                '%-*s ' % (width[i],
                  '%*s' % (row[i].len + (width[i]-row[i].len * j/2), row[i]));
            }.join("");
        }.join("\n") + "\n";
    }
}

func Formatter(text) {
    var textArr = [];
    var widthArr = [];

    text.each_line {
        var words = .split('$');
        textArr.append(words);

        words.each_kv { |i, word|
            if (i == widthArr.len) {
                widthArr.append(word.len);
            }
            elsif (word.len > widthArr[i]) {
                widthArr[i] = word.len;
            }
        }
    }

    return Format(textArr, widthArr);
}

enum |left, middle, right|;
const text = <<'EOT';
Given$a$text$file$of$many$lines,$where$fields$within$a$line$
are$delineated$by$a$single$'dollar'$character,$write$a$program
that$aligns$each$column$of$fields$by$ensuring$that$words$in$each$
column$are$separated$by$at$least$one$space.
Further,$allow$for$each$word$in$a$column$to$be$either$left$
justified,$right$justified,$or$center$justified$within$its$column.
EOT

var f = Formatter(text);

say f.align(left);
say f.align(middle);
say f.align(right);
```

Almkvist-Giullera formula for pi

```
func almkvist_giullera(n) {
    (32 * (14*n * (38*n + 9) + 9) * (6*n)!) / (3 * n!**6)
}

func almkvist_giullera_pi(prec = 70) {

    local Num!PREC = (4*(prec+1)).numify

    var sum = 0
    var target = -1

    for n in (0..Inf) {
        sum += (almkvist_giullera(n) / (10**(6*n + 3)))
        var curr = (sum**-.5).as_dec
        return target if (target == curr)
        target = curr
    }
}

say 'First 10 integer portions: '

10.of {|n|
    say "#{n} #{almkvist_giullera(n)}"
}

with(70) {|n|
    say "π to #{n} decimal places is:"
    say almkvist_giullera_pi(n)
}
```

Output:

```
First 10 integer portions:
0 96
1 5122560
2 190722470400
3 7574824857600000
4 312546150372456000000
5 13207874703225491420651520
6 5672739197930890083292259942400
7 2465060024817298714011276371558400
8 1080657854354639453670407474439566400000
9 477017793915949662874705704908399788800000000
π to 70 decimal places is:
3.14159265358979323846264338327950288419716939937510582097494459230781640
```

Almost prime

```
func is_k_almost_prime(n, k) {
    for (var (p, f) = (2, 0); (f < k) && (p*p <= n); ++p) {
        (n /= p; ++f) while (p `divides` n)
    }
    n > 1 ? (f+1 == k) : (f == k)
}

{ |k|
    var x = 10
    say gather {
        { |i|
            if (is_k_almost_prime(i, k)) {
                take(i)
                --x == 0 && break
            }
        } << 1..Inf
    }
} << 1..5
```

Output:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
[4, 6, 9, 10, 14, 15, 21, 22, 25, 26]
[8, 12, 18, 20, 27, 28, 30, 42, 44, 45]
[16, 24, 36, 40, 54, 56, 60, 81, 84, 88]
[32, 48, 72, 80, 108, 112, 120, 162, 168, 176]
```

Amicable pairs

```
func propdivsum(n) {
    n.sigma - n
}

for i in (1..20000) {
    var j = propdivsum(i)
    say "#{i} #{j}" if (j>i && i==propdivsum(j))
}
```

Output:

```
220 284
1184 1210
2620 2924
5020 5564
6232 6368
10744 10856
12285 14595
17296 18416
```

Anadromes

```
var S = Hash()

File('words.txt').open_r.each {|word|
    word.len > 6 || next
    if (S.has(word.flip)) {
        say "#{word.flip} <=> #{word}"
    }
    else {
        S{word} = true
    }
}
```

Output:

```
amaroid <=> diorama
gateman <=> nametag
deifier <=> reified
leveler <=> relevel
deviler <=> relived
degener <=> reneged
relever <=> reveler
deliver <=> reviled
reliver <=> reviler
redrawer <=> rewarder
dioramas <=> samaroid
revotes <=> setover
sallets <=> stellas
reknits <=> stinker
desserts <=> stressed
pat-pat <=> tap-tap
dessert <=> tressed
```

Anagrams

```
func main(file) {
    file.open_r(\var fh, \var err) ->
        || die "Can't open file `#{file}' for reading: #{err}\n";

    var vls = fh.words.group_by{.sort}.values;
    var max = vls.map{.len}.max;
    vls.grep{.len == max}.each{.join("\t").say};
}

main(%f'/tmp/unixdict.txt');
```

Output:

```
alger   glare   lager   large   regal
abel    able    bale    bela    elba
angel   angle   galen   glean   lange
elan    lane    lean    lena    neal
evil    levi    live    veil    vile
caret   carte   cater   crate   trace
```

Anagrams/Deranged anagrams

```
func find_deranged(Array a) {
    for i in (^a) {
        for j in (i+1 .. a.end) {
            overlaps(a[i], a[j]) || (
                printf("length %d: %s => %s\n", a[i].len, a[i], a[j])
                return true
            )
        }
    }
    return false
}

func main(File file) {

    file.open_r(\var fh, \var err) ->
        || die "Can't open file `#{file}' for reading: #{err}\n"

    var letter_list = Hash()

    # Store anagrams in hash table by letters they contain
    fh.words.each { |word|
        letter_list{word.sort} := [] << word
    }

    letter_list.keys                          \
         .grep {|k| letter_list{k}.len > 1} \     # take only ones with anagrams
         .sort {|a,b| b.len <=> a.len}      \     # sort by length, descending
         .each {|key|

        # If we find a pair, they are the longested due to the sort before
        find_deranged(letter_list{key}) && break
    }
}

main(%f'/tmp/unixdict.txt')
```

Output:

```
length 10: excitation => intoxicate
```

Angle difference between two bearings

```
func bearingAngleDiff(b1, b2) {
    (var b = ((b2 - b1 + 720) % 360)) > 180 ? (b - 360) : b
}

printf("%25s %25s %25s\n", "B1", "B2", "Difference")
printf("%25s %25s %25s\n", "-"*20, "-"*20, "-"*20)


for b1,b2 in ([
                        20,                       45
                       -45,                       45
                       -85,                       90
                       -95,                       90
                       -45,                      125
                       -45,                      145
                   29.4803,                  -88.6381
                  -78.3251,                  -159.036
       -70099.74233810938,         29840.67437876723
       -165313.6666297357,          33693.9894517456
       1174.8380510598456,       -154146.66490124757
        60175.77306795546,         42213.07192354373
    ].slices(2)
) {
    printf("%25s %25s %25s\n", b1, b2, bearingAngleDiff(b1, b2))
}
```

Output:

```
                       B1                        B2                Difference
     --------------------      --------------------      --------------------
                       20                        45                        25
                      -45                        45                        90
                      -85                        90                       175
                      -95                        90                      -175
                      -45                       125                       170
                      -45                       145                      -170
                  29.4803                  -88.6381                  -118.1184
                 -78.3251                  -159.036                   -80.7109
       -70099.74233810938         29840.67437876723          -139.58328312339
       -165313.6666297357          33693.9894517456            -72.3439185187
       1174.8380510598456       -154146.66490124757        -161.5029523074156
        60175.77306795546         42213.07192354373             37.29885558827
```

Animate a pendulum

```
require('Tk')

var root = %O<MainWindow>.new('-title' => 'Pendulum Animation')
var canvas = root.Canvas('-width' => 320, '-height' => 200)

canvas.createLine(  0,  25, 320,  25, '-tags' => <plate>,     '-width' => 2, '-fill' => :grey50)
canvas.createOval(155,  20, 165,  30, '-tags' => <pivot outline>,            '-fill' => :grey50)
canvas.createLine(  1,   1,   1,   1, '-tags' => <rod width>, '-width' => 3, '-fill' => :black)
canvas.createOval(  1,   1,   2,   2, '-tags' => <bob outline>,              '-fill' => :yellow)

canvas.raise(:pivot)
canvas.pack('-fill' => :both, '-expand' => 1)
var(θ = 45, Δθ = 0, length = 150, homeX = 160, homeY = 25)

func show_pendulum() {
    var angle = θ.deg2rad
    var x = (homeX + length*sin(angle))
    var y = (homeY + length*cos(angle))
    canvas.coords(:rod, homeX,  homeY,  x,      y)
    canvas.coords(:bob, x - 15, y - 15, x + 15, y + 15)
}

func recompute_angle() {
    var scaling = 3000/(length**2)

    # first estimate
    var firstΔΔθ = (-sin(θ.deg2rad) * scaling)
    var midΔθ    = (Δθ + firstΔΔθ)
    var midθ     = ((Δθ + midΔθ)/2 + θ)

    # second estimate
    var midΔΔθ = (-sin(midθ.deg2rad) * scaling)
    midΔθ      = ((firstΔΔθ + midΔΔθ)/2 + Δθ)
    midθ       = ((Δθ + midΔθ)/2 + θ)

    # again, first
    midΔΔθ     = (-sin(midθ.deg2rad) * scaling)
    var lastΔθ = (midΔθ + midΔΔθ)
    var lastθ  = ((midΔθ + lastΔθ)/2 + midθ)

    # again, second
    var lastΔΔθ = (-sin(lastθ.deg2rad) * scaling)
    lastΔθ      = ((midΔΔθ + lastΔΔθ)/2 + midΔθ)
    lastθ       = ((midΔθ + lastΔθ)/2 + midθ)

    # Now put the values back in our globals
    Δθ = lastΔθ
    θ  = lastθ
}

func animate(Ref id) {
    recompute_angle()
    show_pendulum()
    *id = root.after(15 => { animate(id) })
}

show_pendulum()
var after_id = root.after(500 => { animate(\after_id) })

canvas.bind('<Destroy>' => { after_id.cancel })
%S<Tk>.MainLoop()
```

Anonymous recursion

__FUNC__ refers to the current function.

```
func fib(n) {
    return NaN if (n < 0)

    func (n) {
        n < 2 ? n
              : (__FUNC__(n-1) + __FUNC__(n-2))
    }(n)
}
```

__BLOCK__ refers to the current block.

```
func fib(n) {
    return NaN if (n < 0)

    {|n|
        n < 2 ? n
              : (__BLOCK__(n-1) + __BLOCK__(n-2))
    }(n)
}
```

Anti-primes

Using the built-in *Number.sigma0* method to count the number of divisors.

```
say with (0) {|max|
    1..Inf -> lazy.grep { (.sigma0 > max) && (max = .sigma0) }.first(20)
}
```

Output:

```
[1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680, 2520, 5040, 7560]
```

Append a record to the end of a text file

```
define (
    RECORD_FIELDS = %w(account password UID GID GECOS directory shell),
    GECOS_FIELDS  = %w(fullname office extension homephone email),
    RECORD_SEP    = ':',
    GECOS_SEP     = ',',
    PASSWD_FILE   = 'passwd.txt',
)

# here's our three records
var records_to_write = [
    Hash(
        account  => 'jsmith',
        password => 'x',
        UID      => 1001,
```

```
        GID      => 1000,
        GECOS    => Hash(
            fullname  => 'John Smith',
            office    => 'Room 1007',
            extension => '(234)555-8917',
            homephone => '(234)555-0077',
            email     => 'jsmith@rosettacode.org',
        ),
        directory => '/home/jsmith',
        shell     => '/bin/bash',
    ),
    Hash(
        account  => 'jdoe',
        password => 'x',
        UID      => 1002,
        GID      => 1000,
        GECOS    => Hash(
            fullname  => 'Jane Doe',
            office    => 'Room 1004',
            extension => '(234)555-8914',
            homephone => '(234)555-0044',
            email     => 'jdoe@rosettacode.org',
        ),
        directory => '/home/jdoe',
        shell     => '/bin/bash',
    ),
];

var record_to_append = Hash(
    account  => 'xyz',
    password => 'x',
    UID      => 1003,
    GID      => 1000,
    GECOS    => Hash(
        fullname  => 'X Yz',
        office    => 'Room 1003',
        extension => '(234)555-8913',
        homephone => '(234)555-0033',
        email     => 'xyz@rosettacode.org',
    ),
    directory => '/home/xyz',
    shell     => '/bin/bash',
);

func record_to_string(rec, sep = RECORD_SEP, fields = RECORD_FIELDS) {
    gather {
        fields.each { |field|
            var r = rec{field} \\ die "Field #{field} not found"
            take(field == 'GECOS' ? record_to_string(r, GECOS_SEP, GECOS_FIELDS)
                                  : r)
        }
    }.join(sep)
}

func write_records_to_file(records, filename = PASSWD_FILE, append = false) {
    File(filename).(append ? :open_a : :open_w)(\var fh, \var err)
    err && die "Can't open #{filename}: #{err}";
    fh.flock(File.LOCK_EX) || die "Can't lock #{filename}: $!"
    fh.seek(0, File.SEEK_END) || die "Can't seek #{filename}: $!"
    records.each { |record| fh.say(record_to_string(record)) }
    fh.flock(File.LOCK_UN) || die "Can't unlock #{filename}: $!"
    fh.close
}

# write two records to file
write_records_to_file(records: records_to_write);
```

```
# append one more record to file
write_records_to_file(records: [record_to_append], append: true);

# test
File(PASSWD_FILE).open_r(\var fh, \var err)
err && die "Can't open file #{PASSWD_FILE}: #{err}"
var lines = fh.lines

# There should be more than one line
assert(lines.len > 1)

# Check the last line
assert_eq(lines[-1], 'xyz:x:1003:1000:X Yz,Room 1003,(234)555-8913,' +
                     '(234)555-0033,xyz@rosettacode.org:/home/xyz:/bin/bash')

say "** Test passed!"
```

Note that flock uses advisory lock; some other program (if it doesn't use flock) can still unexpectedly write to the file.

Append numbers at same position in strings

```
var lists = [
    [1,2,3,4,5,6,7,8,9],
    [10,11,12,13,14,15,16,17,18],
    [19,20,21,22,23,24,25,26,27],
]

say lists.zip.map_2d {|*a| a.join.to_i }
```

Output:

```
[11019, 21120, 31221, 41322, 51423, 61524, 71625, 81726, 91827]
```

Apply a callback to an array

Defining a callback function:

```
func callback(i) { say i**2 }
```

The function will get called for each element:

```
[1,2,3,4].each(callback)
```

Same as above, but with the function inlined:

```
[1,2,3,4].each{|i| say i**2 }
```

For creating a new array, we can use the `Array.map` method:

```
[1,2,3,4,5].map{|i| i**2 }
```

Apply a digital filter (direct form II transposed)

```
func TDF_II_filter(signal, a, b) {
    var out = [0]*signal.len
    for i in ^signal {
        var this = 0
        for j in ^b { i-j >= 0 && (this += b[j]*signal[i-j]) }
        for j in ^a { i-j >= 0 && (this -= a[j]*   out[i-j]) }
        out[i] = this/a[0]
    }
    return out
}

var signal = [
    -0.917843918645,  0.141984778794, 1.20536903482,   0.190286794412,
    -0.662370894973, -1.00700480494, -0.404707073677,  0.800482325044,
     0.743500089861,  1.01090520172,  0.741527555207,  0.277841675195,
     0.400833448236, -0.2085993586,  -0.172842103641, -0.134316096293,
     0.0259303398477, 0.490105989562, 0.549391221511,  0.9047198589
]

var a = [1.00000000, -2.77555756e-16, 3.33333333e-01, -1.85037171e-17]
var b = [0.16666667,  0.5,            0.5,             0.16666667    ]
var f = TDF_II_filter(signal, a, b)

say "["
say f.map { "% 0.8f" % _ }.slices(5).map{.join(', ')}.join(",\n")
say "]"
```

Output:

```
[
-0.15297399, -0.43525783, -0.13604340,  0.69750333,  0.65644469,
-0.43548245, -1.08923946, -0.53767655,  0.51704999,  1.05224975,
 0.96185430,  0.69569009,  0.42435630,  0.19626223, -0.02783512,
-0.21172192, -0.17474556,  0.06925841,  0.38544587,  0.65177084
]
```

Approximate equality

Two values can be compared for approximate equality by using the built-in operator **≅**, available in ASCII as **=~=**, which does approximate comparison by rounding both operands at **(PREC>>2)-1** decimals. However, by default, Sidef uses a floating-point precision of 192 bits.

```
[
    100000000000000.01, 100000000000000.011,
    100.01, 100.011,
    10000000000000.001 / 10000.0, 1000000000.0000001000,
    0.001, 0.0010000001,
    0.00000000000000000000000101, 0.0,
    sqrt(2) * sqrt(2), 2.0,
    -sqrt(2) * sqrt(2), -2.0,
    sqrt(-2) * sqrt(-2), -2.0,
    cbrt(3)**3, 3,
    cbrt(-3)**3, -3,
    100000000000000003.0, 100000000000000004.0,
    3.14159265358979323846, 3.14159265358979324
].each_slice(2, {|a,b|
    say ("#{a} ≅ #{b}: ", a ≅ b)
})
```

Output:

```
100000000000000.01 ≅ 100000000000000.011: false
100.01 ≅ 100.011: false
1000000000.0000001 ≅ 1000000000.0000001: true
0.001 ≅ 0.0010000001: false
0.00000000000000000000000101 ≅ 0: false
2 ≅ 2: true
-2 ≅ -2: true
-2 ≅ -2: true
3 ≅ 3: true
-3-7.829148892683169579692742433456251576313184024l5e-58i ≅ -3: true
100000000000000003 ≅ 100000000000000004: false
3.14159265358979323846 ≅ 3.14159265358979324: false
```

The Number **n.round(-k)** can be used for rounding the number *n* to *k* decimal places. A positive argument can be used for rounding before the decimal point.

```
var a = 100000000000000.01
var b = 100000000000000.011

# Rounding at 2 and 3 decimal places, respectively
say (round(a, -2) == round(b, -2))      # true
say (round(a, -3) == round(b, -3))      # false
```

There is also the built-in **approx_cmp(a, b, k)** method, which is equivalent with **a.round(k) <=> b.round(k)**.

```
var a = 22/7
var b = Num.pi

say ("22/7 ≅ π at 2 decimals: ", approx_cmp(a, b, -2) == 0)
say ("22/7 ≅ π at 3 decimals: ", approx_cmp(a, b, -3) == 0)
```

Output:

```
22/7 ≅ π at 2 decimals: true
22/7 ≅ π at 3 decimals: false
```

Additionally, the **rat_approx** method can be used for computing a very good rational approximation to a given real value:

```
say (1.33333333.rat_approx == 4/3)   # true
say (zeta(-5).rat_approx == -1/252)  # true
```

Rational approximations illustrated for substrings of PI:

```
for k in (3..19) {
    var r = Str(Num.pi).first(k)
    say ("rat_approx(#{r}) = ", Num(r).rat_approx.as_frac)
}
```

Output:

```
rat_approx(3.1) = 31/10
rat_approx(3.14) = 22/7
rat_approx(3.141) = 245/78
rat_approx(3.1415) = 333/106
rat_approx(3.14159) = 355/113
rat_approx(3.141592) = 355/113
rat_approx(3.1415926) = 86953/27678
rat_approx(3.14159265) = 102928/32763
rat_approx(3.141592653) = 103993/33102
rat_approx(3.1415926535) = 1354394/431117
rat_approx(3.14159265358) = 833719/265381
rat_approx(3.141592653589) = 17925491/5705861
rat_approx(3.1415926535897) = 126312511/40206521
rat_approx(3.14159265358979) = 144029661/45846065
rat_approx(3.141592653589793) = 325994779/103767361
rat_approx(3.1415926535897932) = 903259831/287516534
rat_approx(3.14159265358979323) = 1726375805/549522486
```

Arbitrary-precision integers (included)

```
var x = 5**(4**(3**2))
var y = x.to_s
printf("5**4**3**2 = %s...%s and has %i digits\n", y.ft(0,19), y.ft(-20), y.len)
```

Output:

```
5**4**3**2 = 62060698786608744707...92256259918212890625 and has 183231 digits
```

Archimedean spiral

```
require('Imager')
define π = Num.pi

var (w, h) = (400, 400)
var img = %O<Imager>.new(xsize => w, ysize => h)

for 0 in (0 .. 52*π -> by(0.025)) {
    img.setpixel(
        x => floor(cos(0 / π)*0 + w/2),
        y => floor(sin(0 / π)*0 + h/2),
        color => [255, 0, 0]
    )
}

img.write(file => 'Archimedean_spiral.png')
```

Output image

Arithmetic/Complex

```
var a = 1:1                 # Complex(1, 1)
var b = 3.14159:1.25        # Complex(3.14159, 1.25)

[   a + b,                  # addition
    a * b,                  # multiplication
    -a,                     # negation
    a.inv,                  # multiplicative inverse
    a.conj,                 # complex conjugate
    a.abs,                  # abs
    a.sqrt,                 # sqrt
    b.re,                   # real
    b.im,                   # imaginary
].each { |c| say c }
```

Output:

```
4.14159+2.25i
1.89159+4.39159i
-1-i
0.5-0.5i
1-i
1.414213562373095048801688724209 7
1.098684113467809966039801195240 68+0.455089860562227341304357757822 47i
3.14159
1.25
```

Arithmetic/Integer

```
var a = read("First number: ", Number).int
var b = read("Second number: ", Number).int

%w'+ - * // % ** ^ | & << >>'.each { |op|
    "#{a} #{op} #{b} = #{a.$op(b)}".say
}
```

Output:

```
First number: 1234
Second number: 7
1234 + 7 = 1241
1234 - 7 = 1227
1234 * 7 = 8638
1234 // 7 = 176
1234 % 7 = 2
1234 ** 7 = 4357186184021382204544
1234 ^ 7 = 1237
1234 | 7 = 1239
1234 & 7 = 2
1234 << 7 = 157952
1234 >> 7 = 9
```

Arithmetic/Rational

Sidef has built-in support for rational numbers.

```
for n in (1 .. 2**19) {
    var frac = 0

    n.divisors.each {|d|
        frac += 1/d
    }

    if (frac.is_int) {
        say "Sum of reciprocal divisors of #{n} = #{frac} exactly #{
            frac == 2 ? '- perfect!' : ''
        }"
    }
}
```

Output:

```
Sum of reciprocal divisors of 1 = 1 exactly
Sum of reciprocal divisors of 6 = 2 exactly - perfect!
Sum of reciprocal divisors of 28 = 2 exactly - perfect!
Sum of reciprocal divisors of 120 = 3 exactly
Sum of reciprocal divisors of 496 = 2 exactly - perfect!
Sum of reciprocal divisors of 672 = 3 exactly
Sum of reciprocal divisors of 8128 = 2 exactly - perfect!
Sum of reciprocal divisors of 30240 = 4 exactly
Sum of reciprocal divisors of 32760 = 4 exactly
Sum of reciprocal divisors of 523776 = 3 exactly
```

Arithmetic-geometric mean

```
func agm(a, g) {
    loop {
        var (a1, g1) = ((a+g)/2, sqrt(a*g))
        [a1,g1] == [a,g] && return a
        (a, g) = (a1, g1)
    }
}

say agm(1, 1/sqrt(2))
```

Output:

```
0.847213084793979086606499123482191636 48
```

Arithmetic-geometric mean/Calculate Pi

```
func agm_pi(digits) {
    var acc = (digits + 8);

    local Num!PREC = 4*digits;

    var an = 1;
    var bn = sqrt(0.5);
    var tn = 0.5**2;
    var pn = 1;

    while (pn < acc) {
        var prev_an = an;
        an = (bn+an / 2);
        bn = sqrt(bn * prev_an);
        prev_an -= an;
        tn -= (pn * prev_an**2);
        pn *= 2;
    }

    ((an+bn)**2 / 4*tn).to_s
}

say agm_pi(100);
```

Output:

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706 8
```

Arithmetic coding/As a generalized change of radix

```
func cumulative_freq(freq) {
    var cf = Hash()
    var total = 0
    256.range.each { |b|
        if (freq.contains(b)) {
            cf{b} = total
```

```
            total += freq{b}
        }
    }
    return cf
}

func arithmethic_coding(bytes, radix=10) {

    # The frequency characters
    var freq = Hash()
    bytes.each { |b| freq{b} := 0 ++ }

    # The cumulative frequency table
    var cf = cumulative_freq(freq)

    # Base
    var base = bytes.len

    # Lower bound
    var L = 0

    # Product of all frequencies
    var pf = 1

    # Each term is multiplied by the product of the
    # frequencies of all previously occurring symbols
    bytes.each { |b|
        var x = cf{b}
        L *= base += x*pf
        pf *= freq{b}
    }

    # Upper bound
    var U = L+pf

    var pow = pf.log(radix).int
    var enc = ((U-1) // radix**pow)

    return (enc, pow, freq)
}

func arithmethic_decoding(enc, radix, pow, freq) {

    # Multiply enc by radix^pow
    enc *= radix**pow;

    var base = 0
    freq.each_value { |v| base += v }

    # Create the cumulative frequency table
    var cf = cumulative_freq(freq);

    # Create the dictionary
    var dict = Hash()
    cf.each_kv { |k,v|
        dict{v} = k
    }

    # Fill the gaps in the dictionary
    var lchar = ''
    base.range.each { |i|
        if (dict.contains(i)) {
            lchar = dict{i}
        }
        elsif (!lchar.is_empty) {
```

```
            dict{i} = lchar
        }
    }

    # Decode the input number
    var decoded = []
    base.range.reverse.each { |i|

        var pow = base**i;
        var div = enc//pow

        var c  = dict{div}
        var fv = freq{c}
        var cv = cf{c}

        var rem = ((enc - pow*cv) // fv)

        enc = rem
        decoded << c
    }

    # Return the decoded output
    return decoded
}

var radix = 10;      # can be any integer greater or equal with 2

%w(DABDDB DABDDBBDDBA ABRACADABRA TOBEORNOTTOBEORTOBEORNOT).each { |str|

    var (enc, pow, freq) = arithmethic_coding(str.bytes, radix)
    var dec = arithmethic_decoding(enc, radix, pow, freq).join_bytes('UTF-8')

    printf("%-25s=> %19s * %d^%s\n", str, enc, radix, pow);

    if (str != dec) {
        die "\tHowever that is incorrect!"
    }
}
```

Output:

```
DABDDB                    =>                 251 * 10^2
DABDDBBDDBA               =>              167351 * 10^6
ABRACADABRA               =>             7954170 * 10^4
TOBEORNOTTOBEORTOBEORNOT => 1150764267498783364 * 10^15
```

Arithmetic evaluation

```
func evalArithmeticExp(s) {

    func evalExp(s) {

        func operate(s, op) {
           s.split(op).map{|c| Number(c) }.reduce(op)
        }

        func add(s) {
            operate(s.sub(/^\+/,'').sub(/\++/,'+'), '+')
        }

        func subtract(s) {
            s.gsub!(/(\+-|-\+)/,'-')

            if (s ~~ /--/) {
                return(add(s.sub(/--/,'+')))
            }

            var b = s.split('-')
            b.len == 3 ? (-1*Number(b[1]) - Number(b[2]))
                       : operate(s, '-')
        }

        s.gsub!(/[()]/,'').gsub!(/-\+/, '-')

        var reM  = /\*/
        var reMD = %r"(\d+\.?\d*\s*[*/]\s*[+-]?\d+\.?\d*)"

        var reA  = /\d\+/
        var reAS = /(-?\d+\.?\d*\s*[+-]\s*[+-]?\d+\.?\d*)/

        while (var match = reMD.match(s)) {
            match[0] ~~ reM
                ? s.sub!(reMD, operate(match[0], '*').to_s)
                : s.sub!(reMD, operate(match[0], '/').to_s)
        }

        while (var match = reAS.match(s)) {
            match[0] ~~ reA
                ? s.sub!(reAS,      add(match[0]).to_s)
                : s.sub!(reAS, subtract(match[0]).to_s)
        }

        return s
    }

    var rePara = /(\([^\(\)]*\))/
    s.split!.join!('').sub!(/^\+/,'')

    while (var match = s.match(rePara)) {
        s.sub!(rePara, evalExp(match[0]))
    }

    return Number(evalExp(s))
}
```

Testing the function:

```
for expr,res in [
     ['2+3'                                        =>        5],
     ['-4-3'                                       =>       -7],
     ['-+2+3/4'                                    =>    -1.25],
     ['2*3-4'                                      =>        2],
     ['2*(3+4)+2/4'                                => 2/4 + 14],
     ['2*-3--4+-0.25'                               =>    -2.25],
     ['2 * (3 + (4 * 5 + (6 * 7) * 8) - 9) * 10' =>     7000],
] {
    var num = evalArithmeticExp(expr)
    assert_eq(num, res)
    "%-45s == %10g\n".printf(expr, num)
}
```

Array concatenation

```
var arr1 = [1, 2, 3];
var arr2 = [4, 5, 6];
var arr3 = (arr1 + arr2);   # => [1, 2, 3, 4, 5, 6]
```

Array length

```
var arr = ['apple', 'orange'];
say arr.len;        #=> 2
say arr.end;        #=> 1 (zero based)
```

Array search

```
struct City {
    String name,
    Number population,
}

var cities = [
    City("Lagos", 21),
    City("Cairo", 15.2),
    City("Kinshasa-Brazzaville", 11.3),
    City("Greater Johannesburg", 7.55),
    City("Mogadishu", 5.85),
    City("Khartoum-Omdurman", 4.98),
    City("Dar Es Salaam", 4.7),
    City("Alexandria", 4.58),
    City("Abidjan", 4.4),
    City("Casablanca", 3.98),
]

say cities.index{|city| city.name == "Dar Es Salaam"}   # => 6
say cities.first{|city| city.population < 5.0}.name     # => Khartoum-Omdurman
```

Arrays

```
# create an empty array
var arr = []

# push objects into the array
arr << "a"            #: ['a']
arr.append(1,2,3)     #: ['a', 1, 2, 3]

# change an element inside the array
arr[2] = "b"          #: ['a', 1, 'b', 3]

# set the value at a specific index in the array (with autovivification)
arr[5] = "end"        #: ['a', 1, 'b', 3, nil, 'end']

# resize the array
arr.resize_to(-1)     #: []

# slice assignment
arr[0..2] = ('a'..'c')...       #: ['a', 'b', 'c']

# indices as arrays
var indices = [0, -1]
arr[indices] = ("foo", "baz")   #: ['foo', 'b', 'baz']

# retrieve multiple elements
var *elems = arr[0, -1]
say elems               #=> ['foo', 'baz']

# retrieve an element
say arr[-1]             #=> 'baz'
```

Ascending primes

```
func primes_with_ascending_digits(base = 10) {

    var list = []
    var digits = @(1..^base -> flip)

    var end_digits = digits.grep { .is_coprime(base) }
    list << digits.grep { .is_prime && !.is_coprime(base) }...

    for k in (0 .. digits.end) {
        digits.combinations(k, {|*a|
            var v = a.digits2num(base)
            end_digits.each {|d|
                var n = (v*base + d)
                next if ((n >= base) && (a[0] >= d))
                list << n if (n.is_prime)
            }
        })
    }

    list.sort
}

var arr = primes_with_ascending_digits()

say "There are #{arr.len} ascending primes.\n"

arr.each_slice(10, {|*a|
    say a.map { '%8s' % _ }.join(' ')
})
```

Output:

```
There are 100 ascending primes.

       2        3        5        7       13       17       19       23       29       37
      47       59       67       79       89      127      137      139      149      157
     167      179      239      257      269      347      349      359      367      379
     389      457      467      479      569     1237     1249     1259     1279     1289
    1367     1459     1489     1567     1579     1789     2347     2357     2389     2459
    2467     2579     2689     2789     3457     3467     3469     4567     4679     4789
    5689    12347    12379    12457    12479    12569    12589    12689    13457    13469
   13567    13679    13789    15679    23459    23567    23689    23789    25679    34589
   34679   123457   123479   124567   124679   125789   134789   145679   234589   235679
  235789   245789   345679   345689  1234789  1235789  1245689  1456789 12356789 23456789
```

Assertions

```
var num = pick(0..100);
assert_eq(num, 42);         # dies when "num" is not 42
```

Output:

```
assert_eq: 26 == 42 is false at assertions.sf line 2.
```

Associative array/Creation

```
var hash = Hash(
    key1 => 'value1',
    key2 => 'value2',
)

# Add a new key-value pair
hash{:key3} = 'value3'
```

Associative array/Iteration

```
var hash = Hash(
    key1 => 'value1',
    key2 => 'value2',
)

# Iterate over key-value pairs
hash.each { |key, value|
    say "#{key}: #{value}"
}

# Iterate only over keys
hash.keys.each { |key|
    say key
}

# Iterate only over values
hash.values.each { |value|
    say value
}
```

Output:

```
key1: value1
key2: value2
key1
key2
value1
value2
```

Attractive numbers

```
func is_attractive(n) {
    n.bigomega.is_prime
}

1..120 -> grep(is_attractive).say
```

Output:

```
[4, 6, 8, 9, 10, 12, 14, 15, 18, 20, 21, 22, 25, 26, 27, 28, 30, 32, 33, 34, 35, 38, 39, 42, 44, 45, 46
```

Average loop length

```
func find_loop(n) {
    var seen = Hash()
    loop {
        with (irand(1, n)) { |r|
            seen.has(r) ? (return seen.len) : (seen{r} = true)
        }
    }
}

print " N    empiric      theoric      (error)\n";
print "===  =========  ============  =========\n";

define MAX    = 20
define TRIALS = 1000

for n in (1..MAX) {
    var empiric = (1..TRIALS -> sum { find_loop(n) } / TRIALS)
    var theoric = (1..n -> sum {|k| prod(n - k + 1 .. n) * k**2 / n**(k+1) })

    printf("%3d  %9.4f  %12.4f   (%5.2f%%)\n",
        n, empiric, theoric, 100*(empiric-theoric)/theoric)
}
```

Output:

```
 N    empiric      theoric      (error)
===  =========  ============  =========
  1     1.0000        1.0000   ( 0.00%)
  2     1.4990        1.5000   (-0.07%)
  3     1.8560        1.8889   (-1.74%)
  4     2.1730        2.2188   (-2.06%)
  5     2.5440        2.5104   ( 1.34%)
  6     2.7490        2.7747   (-0.93%)
  7     3.0390        3.0181   ( 0.69%)
  8     3.1820        3.2450   (-1.94%)
  9     3.4520        3.4583   (-0.18%)
 10     3.6580        3.6602   (-0.06%)
 11     3.9650        3.8524   ( 2.92%)
 12     3.9920        4.0361   (-1.09%)
 13     4.1270        4.2123   (-2.03%)
 14     4.3710        4.3820   (-0.25%)
 15     4.5520        4.5458   ( 0.14%)
 16     4.7120        4.7043   ( 0.16%)
 17     4.9400        4.8579   ( 1.69%)
 18     5.0070        5.0071   (-0.00%)
 19     5.2370        5.1522   ( 1.65%)
 20     5.2940        5.2936   ( 0.01%)
```

Averages/Arithmetic mean

```
func avg(Array list) {
    list.len > 0 || return 0
    list.sum / list.len
}

say avg([Inf, Inf])
say avg([3,1,4,1,5,9])
say avg([1e+20, 3, 1, 4, 1, 5, 9, -1e+20])
say avg([10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 0, 0, 0, 0.11])
say avg([10, 20, 30, 40, 50, -100, 4.7, -1100])
```

Output:

```
Inf
3.83333333333333333333333333333333
2.875
3.674
-130.6625
```

Averages/Mean angle

```
func mean_angle(angles) {
    atan2(
        Math.avg(angles.map{ .deg2rad.sin }...),
        Math.avg(angles.map{ .deg2rad.cos }...),
    ) -> rad2deg;
}

[[350,10], [90,180,270,360], [10,20,30]].each { |angles|
  say "The mean angle of #{angles.dump} is: #{ '%.2f' % mean_angle(angles)} degrees";
}
```

Output:

```
The mean angle of [350, 10] is: 0.00 degrees
The mean angle of [90, 180, 270, 360] is: -90.00 degrees
The mean angle of [10, 20, 30] is: 20.00 degrees
```

Averages/Mean time of day

Using the *mean_angle()* function from: "Averages/Mean angle"

```
func time2deg(t) {
  (var m = t.match(/^(\d\d):(\d\d):(\d\d)$/)) || die "invalid time"
  var (hh,mm,ss) = m.cap.map{.to_i}...
  ((hh ~~ 24.range) && (mm ~~ 60.range) && (ss ~~ 60.range)) || die "invalid time"
  (hh*3600 + mm*60 + ss) * 360 / 86400
}

func deg2time(d) {
  var sec = ((d % 360) * 86400 / 360)
  "%02d:%02d:%02d" % (sec/3600, (sec%3600)/60, sec%60)
}

func mean_time(times) {
  deg2time(mean_angle(times.map {|t| time2deg(t)}))
}

say mean_time(["23:00:17", "23:40:20", "00:12:45", "00:17:19"])
```

Output:

```
23:47:43
```

Averages/Median

```
func median(arry) {
    var srtd = arry.sort;
    var alen = srtd.length;
    srtd[(alen-1)/2]+srtd[alen/2] / 2;
}
```

Averages/Mode

```
func mode(array) {
    var c = Hash()
    array.each{|i| c{i} := 0 ++}
    var max = c.values.max
    c.keys.grep{|i| c{i} == max}
}
```

Calling the function

```
say mode([1, 3, 6, 6, 6, 6, 7, 7, 12, 12, 17]).join(' ')
say mode([1, 1, 2, 4, 4]).join(' ')
```

Output:

```
6
1 4
```

If you just want one mode (instead of all of them), here's a one-liner for that:

```
func one_mode(arr) {
    arr.max_by{|i| arr.count(i)}
}
```

Averages/Pythagorean means

```
func A(a) { a.sum / a.len }
func G(a) { a.prod.root(a.len) }
func H(a) { a.len / a.map{1/_}.sum }
```

The same thing, using hyper-operators:

```
func A(a) { a«+» / a.len }
func G(a) { a«*» ** (1/a.len) }
func H(a) { a.len / (a«/«1 «+») }
```

Calling the functions:

```
say("A(1,...,10) = ", A(1..10))
say("G(1,...,10) = ", G(1..10))
say("H(1,...,10) = ", H(1..10))
```

Output:

```
A(1,...,10) = 5.5
G(1,...,10) = 4.52872868811676476220330933719550879349 9
H(1,...,10) = 3.414171521474055006096734859775098225173
```

Averages/Root mean square

```
func rms(a) {
    sqrt(a.map{.**2}.sum / a.len)
}

say rms(1..10)
```

Output:

```
6.204836822995428298066620977724737849927965295 36
```

Averages/Simple moving average

Implemented with closures:

```
func simple_moving_average(period) {

    var list = []
    var sum = 0

    func (number) {
        list.append(number)
        sum += number
        if (list.len > period) {
            sum -= list.shift
        }
        (sum / list.length)
    }
}

var ma3 = simple_moving_average(3)
var ma5 = simple_moving_average(5)

for num (1..5, flip(1..5)) {
  printf("Next number = %d, SMA_3 = %.3f, SMA_5 = %.1f\n",
    num, ma3.call(num), ma5.call(num))
}
```

Implemented as a class:

```
class sma_generator(period, list=[], sum=0) {

    method SMA(number) {
        list.append(number)
        sum += number
        if (list.len > period) {
            sum -= list.shift
        }
        (sum / list.len)
    }
}

var ma3 = sma_generator(3)
var ma5 = sma_generator(5)

for num (1..5, flip(1..5)) {
  printf("Next number = %d, SMA_3 = %.3f, SMA_5 = %.1f\n",
    num, ma3.SMA(num), ma5.SMA(num))
}
```

Output:

```
Next number = 1, SMA_3 = 1.000, SMA_5 = 1.0
Next number = 2, SMA_3 = 1.500, SMA_5 = 1.5
Next number = 3, SMA_3 = 2.000, SMA_5 = 2.0
Next number = 4, SMA_3 = 3.000, SMA_5 = 2.5
Next number = 5, SMA_3 = 4.000, SMA_5 = 3.0
Next number = 5, SMA_3 = 4.667, SMA_5 = 3.8
Next number = 4, SMA_3 = 4.667, SMA_5 = 4.2
Next number = 3, SMA_3 = 4.000, SMA_5 = 4.2
Next number = 2, SMA_3 = 3.000, SMA_5 = 3.8
Next number = 1, SMA_3 = 2.000, SMA_5 = 3.0
```

AVL tree

```
class AVLtree {

    has root = nil

    struct Node {
        Number key,
        Number balance = 0,
        Node left = nil,
        Node right = nil,
        Node parent = nil,
    }

    method insert(key) {
        if (root == nil) {
            root = Node(key)
            return true
        }

        var n = root
        var parent = nil

        loop {
            if (n.key == key) {
                return false
            }
            parent = n
            var goLeft = (n.key > key)
            n = (goLeft ? n.left : n.right)

            if (n == nil) {
                var tn = Node(key, parent: parent)
                if (goLeft) {
                    parent.left = tn
                }
                else {
                    parent.right = tn
                }
                self.rebalance(parent)
                break
            }
        }

        return true
    }

    method delete_key(delKey) {
        if (root == nil) { return nil }

        var n = root
        var parent = root
        var delNode = nil
        var child = root

        while (child != nil) {
            parent = n
            n = child
            child = (delKey >= n.key ? n.right : n.left)
            if (delKey == n.key) {
                delNode = n
            }
        }

        if (delNode != nil) {
```

```
            delNode.key = n.key
            child = (n.left != nil ? n.left : n.right)

            if (root.key == delKey) {
                root = child
            }
            else {
                if (parent.left == n) {
                    parent.left = child
                }
                else {
                    parent.right = child
                }
                self.rebalance(parent)
            }
        }
    }

    method rebalance(n) {
        if (n == nil) { return nil }
        self.setBalance(n)

        given (n.balance) {
            when (-2) {
                if (self.height(n.left.left) >= self.height(n.left.right)) {
                    n = self.rotate(n, :right)
                }
                else {
                    n = self.rotate_twice(n, :left, :right)
                }
            }
            when (2) {
                if (self.height(n.right.right) >= self.height(n.right.left)) {
                    n = self.rotate(n, :left)
                }
                else {
                    n = self.rotate_twice(n, :right, :left)
                }
            }
        }

        if (n.parent != nil) {
            self.rebalance(n.parent)
        }
        else {
            root = n
        }
    }

    method rotate(a, dir) {
        var b = (dir == :left ? a.right : a.left)
        b.parent = a.parent

        (dir == :left) ? (a.right = b.left)
                       : (a.left  = b.right)

        if (a.right != nil) {
            a.right.parent = a
        }

        b.$dir = a
        a.parent = b

        if (b.parent != nil) {
            if (b.parent.right == a) {
                b.parent.right = b
```

```
            }
            else {
                b.parent.left = b
            }
        }

        self.setBalance(a, b)
        return b
    }

    method rotate_twice(n, dir1, dir2) {
        n.left = self.rotate(n.left, dir1)
        self.rotate(n, dir2)
    }

    method height(n) {
        if (n == nil) { return -1 }
        1 + Math.max(self.height(n.left), self.height(n.right))
    }

    method setBalance(*nodes) {
        nodes.each { |n|
            n.balance = (self.height(n.right) - self.height(n.left))
        }
    }

    method printBalance {
        self.printBalance(root)
    }

    method printBalance(n) {
        if (n != nil) {
            self.printBalance(n.left)
            print(n.balance, ' ')
            self.printBalance(n.right)
        }
    }
}

var tree = AVLtree()

say "Inserting values 1 to 10"
{|i| tree.insert(i) } << 1..10
print "Printing balance: "
tree.printBalance
```

Output:

```
Inserting values 1 to 10
Printing balance: 0 0 0 1 0 0 0 0 1 0
```

Babbage problem

```
var n = 0
while (n*n % 1000000 != 269696) {
    n += 2
}
say n
```

Output:

```
25264
```

Balanced brackets

```
func balanced (str) {

    var depth = 0
    str.each { |c|
           if(c=='['){ ++depth }
        elsif(c==']'){ --depth < 0 && return false }
    }

    return !depth
}

for str [']','[','[[]',']][]','[[]]','[[]]]][][]]','x[ y [ [] z ]][ 1 ][]abcd'] {
    printf("%sbalanced\t: %s\n", balanced(str) ? "" : "NOT ", str)
}
```

Output:

```
NOT balanced    : ]
NOT balanced    : [
NOT balanced    : [[]
NOT balanced    : ][]
balanced        : [[]]
NOT balanced    : [[]]]][][]]
balanced        : x[ y [ [] z ]][ 1 ][]abcd
```

Barnsley fern

```
require('Imager')

var w = 640
var h = 640

var img = %O<Imager>.new(xsize => w, ysize => h, channels => 3)
var green = %O<Imager::Color>.new('#00FF00')

var (x, y) = (0.float, 0.float)

for r in (^1e4 -> lazy.map { 100.irand }) {
  (x, y) = (
    if    (r <=  1) { ( 0.00*x - 0.00*y,  0.00*x + 0.16*y + 0.00) }
    elsif (r <=  8) { ( 0.20*x - 0.26*y,  0.23*x + 0.22*y + 1.60) }
    elsif (r <= 15) { (-0.15*x + 0.28*y,  0.26*x + 0.24*y + 0.44) }
    else            { ( 0.85*x + 0.04*y, -0.04*x + 0.85*y + 1.60) }
  )
  img.setpixel(x => w/2 + 60*x, y => 60*y, color => green)
}

img.flip(dir => 'v')
img.write(file => 'barnsleyFern.png')
```

Output image

Base58Check encoding

```
func encode_base58(n) {
    static chars = %w(
        1 2 3 4 5 6 7 8 9
        A B C D E F G H   J K L M N   P Q R S T U V W X Y Z
        a b c d e f g h i j k   m n o p q r s t u v w x y z
    )
    [chars[n.polymod(n.ilog(58).of(58)...)]].join.flip
}

var tests = [
    [25420294593250030202636073700053352635053786165627414518, "6UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM"],
    [97, "2g"],[6447714, "a3gV"],[6513507, "aPEr"],
    [658885050385564465925592505944209249682185612903, "2cFupjhnEsSn59qHXstmK2ffpLv2"],
    [349694840079, "ABnLTmg"], [3529059230209907258589, "3SEo3LWLoPntC"],
    [1462650772, "3EFU7m"], [1117661258925082241147681, "EJDM8drfXA6uyA"], [281563422, "Rt5zm"]
]

for num, enc in (tests) {
    printf("%56s -> %s\n", num, encode_base58(num))
    assert_eq(encode_base58(num), enc)
}
```

Output:

```
25420294593250030202636073700053352635053786165627414518 -> 6UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM
                                                        97 -> 2g
                                                   6447714 -> a3gV
                                                   6513507 -> aPEr
        6588850503855644659255925059442092496821856129O3 -> 2cFupjhnEsSn59qHXstmK2ffpLv2
                                              349694840079 -> ABnLTmg
                                   3529059230209907258589 -> 3SEo3LWLoPntC
                                               1462650772 -> 3EFU7m
                                1117661258925082241147681 -> EJDM8drfXA6uyA
                                               281563422 -> Rt5zm
```

Base64 decode data

```
var data = <<'EOT'
VG8gZXJyIGlzIGh1bWFuLCBidXQgdG8gcmVhbGx5IGZvdWwgdGhpbmdzIHVwIHlvdSBuZWVkIGEgY2
9tcHV0ZXIuCiAgICAtLSBQYXVsIFIuIEVocmxpY2g=
EOT

say data.decode_base64
```

Output:

```
To err is human, but to really foul things up you need a computer.
    -- Paul R. Ehrlich
```

Base64 encode data

```
var data = %f'favicon.ico'.read(:raw)   # binary string
print data.encode_base64                # print to STDOUT
```

Base 16 numbers needing a to f

```
(0..500).grep {|n| n.digits(16).any {|d| d >= 10} }.join(" ").say
```

Output:

```
10 11 12 13 14 15 26 27 28 29 30 31 42 43 44 45 46 47 58 59 60 61 62 63 74 75 76 77 78 79 90 91 92 93 9
```

Bell numbers

Built-in:

```
say 15.of { .bell }
```

Formula as a sum of Stirling numbers of the second kind:

```
func bell(n) { sum(0..n, {|k| stirling2(n, k) }) }
```

Via Aitken's array (optimized for space):

```
func bell_numbers (n) {

    var acc = []
    var bell = [1]

    (n-1).times {
        acc.unshift(bell[-1])
        acc.accumulate!
        bell.push(acc[-1])
    }

    bell
}

var B = bell_numbers(50)
say "The first 15 Bell numbers: #{B.first(15).join(', ')}"
say "The fiftieth Bell number : #{B[50-1]}"
```

Output:

```
The first 15 Bell numbers: 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437
The fiftieth Bell number : 10726137154573358400342215518590002633917247281
```

Aitken's array:

```
func aitken_array (n) {

    var A = [1]

    [[1]] + (n-1).of {
        A = [A[-1], A...].accumulate
    }
}

aitken_array(10).each { .say }
```

Output:

```
[1]
[1, 2]
[2, 3, 5]
[5, 7, 10, 15]
[15, 20, 27, 37, 52]
[52, 67, 87, 114, 151, 203]
[203, 255, 322, 409, 523, 674, 877]
[877, 1080, 1335, 1657, 2066, 2589, 3263, 4140]
[4140, 5017, 6097, 7432, 9089, 11155, 13744, 17007, 21147]
[21147, 25287, 30304, 36401, 43833, 52922, 64077, 77821, 94828, 115975]
```

Aitken's array (recursive definition):

```
func A((0), (0))       { 1 }
func A(n, (0))         { A(n-1, n-1) }
func A(n, k) is cached { A(n, k-1) + A(n-1, k-1) }

for n in (^10) {
    say (0..n -> map{|k| A(n, k) })
}
```

(same output as above)

Benford's law

```
var (actuals, expected) = ([], [])
var fibonacci = 1000.of {|i| fib(i).digit(-1) }

for i (1..9) {
    var num = fibonacci.count_by {|j| j == i }
    actuals.append(num / 1000)
    expected.append(1 + (1/i) -> log10)
}

"%17s%17s\n".printf("Observed","Expected")

for i (1..9) {
    "%d : %11s %%%15s %%\n".printf(
            i, "%.2f".sprintf(100 *  actuals[i - 1]),
               "%.2f".sprintf(100 * expected[i - 1]),
    )
}
```

Output:

```
          Observed         Expected
1 :       30.10 %          30.10 %
2 :       17.70 %          17.61 %
3 :       12.50 %          12.49 %
4 :        9.50 %           9.69 %
5 :        8.00 %           7.92 %
6 :        6.70 %           6.69 %
7 :        5.60 %           5.80 %
8 :        5.30 %           5.12 %
9 :        4.50 %           4.58 %
```

Bernoulli numbers

Built-in:

```
say bernoulli(42).as_frac    #=> 1520097643918070802691/1806
```

Recursive solution (with auto-memoization):

```
func bernoulli_number(n) is cached {

    n.is_one && return 1/2
    n.is_odd && return   0

    1 - sum(^n, {|k|
        binomial(n,k) * __FUNC__(k) / (n - k + 1)
    })
}

for n in (0..60) {
    var Bn = bernoulli_number(n) || next
    printf("B(%2d) = %44s / %s\n", n, Bn.nude)
}
```

Using Ramanujan's congruences (pretty fast):

```
func ramanujan_bernoulli_number(n) is cached {

    return 1/2 if n.is_one
    return 0   if n.is_odd

    ((n%6 == 4 ? -1/2 : 1) * (n+3)/3 - sum(1 .. (n - n%6)/6, {|k|
        binomial(n+3, n - 6*k) * __FUNC__(n - 6*k)
    })) / binomial(n+3, n)
}
```

Using Euler's product formula for the Riemann zeta function and the Von Staudt–Clausen theorem (very fast):

```
func bernoulli_number_from_zeta(n) {

    n.is_zero && return   1
    n.is_one  && return 1/2
    n.is_odd  && return   0

    var log2B = (log(4*Num.tau*n)/2 + n*log(n) - n*log(Num.tau) - n)/log(2)
    local Num!PREC = *(int(n + log2B) + (n <= 90 ? 18 : 0))

    var K = 2*(n! / Num.tau**n)
    var d = n.divisors.grep {|k| is_prime(k+1) }.prod {|k| k+1 }
    var z = ceil((K*d).root(n-1)).primes.prod {|p| 1 - p.float**(-n) }

    (-1)**(n/2 + 1) * int(ceil(d*K / z)) / d
}
```

The Akiyama–Tanigawa algorithm:

```
func bernoulli_print {
    var a = []
    for m in (0..60) {
        a << 1/(m+1)
        for j in (1..m -> flip) {
            (a[j-1] -= a[j]) *= j
        }
        a[0] || next
        printf("B(%2d) = %44s / %s\n", m, a[0].nude)
    }
}

bernoulli_print()
```

Output:

```
B( 0) =                                            1 / 1
B( 1) =                                            1 / 2
B( 2) =                                            1 / 6
B( 4) =                                           -1 / 30
B( 6) =                                            1 / 42
B( 8) =                                           -1 / 30
B(10) =                                            5 / 66
B(12) =                                         -691 / 2730
B(14) =                                            7 / 6
B(16) =                                        -3617 / 510
B(18) =                                        43867 / 798
B(20) =                                      -174611 / 330
B(22) =                                       854513 / 138
B(24) =                                   -236364091 / 2730
B(26) =                                      8553103 / 6
B(28) =                                 -23749461029 / 870
B(30) =                                8615841276005 / 14322
B(32) =                               -7709321041217 / 510
B(34) =                                2577687858367 / 6
B(36) =                        -26315271553053477373 / 1919190
B(38) =                             2929993913841559 / 6
B(40) =                       -261082718496449122051 / 13530
B(42) =                       1520097643918070802691 / 1806
B(44) =                     -27833269579301024235023 / 690
B(46) =                     596451111593912163277961 / 282
B(48) =                -5609403368997817686249127547 / 46410
B(50) =                  495057205241079648212477525 / 66
B(52) =              -801165718135489957347924991853 / 1590
B(54) =             29149963634884862421418123812691 / 798
B(56) =           -2479392929313226753685415739663229 / 870
B(58) =          84483613348880041862046775994036021 / 354
B(60) = -1215233140483755572040304994079820246041491 / 56786730
```

Best shuffle

```
func best_shuffle(String orig) -> (String, Number) {

    var s = orig.chars
    var t = s.shuffle

    for i (^s) {
        for j (^s) {
            if (i!=j && t[i]!=s[j] && t[j]!=s[i]) {
                t[i, j] = t[j, i]
                break
            }
        }
    }

    (t.join, s ~Z== t -> count(true))
}

for word (<abracadabra seesaw elk grrrrrr up a>) {
    var (sword, score) = best_shuffle(word)
    "%-12s %12s: %d\n".printf(word, sword, score)
}
```

Output:

```
abracadabra   daabacarrab: 0
seesaw             esaews: 0
elk                   lke: 0
grrrrrr           rgrrrrr: 5
up                     pu: 0
a                       a: 1
```

Bilinear interpolation

```
require('Imager')

func scale(img, scaleX, scaleY) {
    var (width, height) = (img.getwidth, img.getheight)
    var (newWidth, newHeight) = (int(width*scaleX), int(height*scaleY))

    var out = %O<Imager>.new(xsize => newWidth, ysize => newHeight)

    var lerp = { |s, e, t|
        s + t*(e-s)
    }

    var blerp = { |c00, c10, c01, c11, tx, ty|
        lerp(lerp(c00, c10, tx), lerp(c01, c11, tx), ty)
    }

    for x,y in (^newWidth ~X ^newHeight) {
        var gxf = (x/newWidth  * (width  - 1))
        var gyf = (y/newHeight * (height - 1))

        var gx = gxf.int
        var gy = gyf.int

        var *c00 = img.getpixel(x => gx,   y => gy  ).rgba
        var *c10 = img.getpixel(x => gx+1, y => gy  ).rgba
        var *c01 = img.getpixel(x => gx,   y => gy+1).rgba
        var *c11 = img.getpixel(x => gx+1, y => gy+1).rgba

        var rgb = 3.of { |i|
            blerp(c00[i], c10[i], c01[i], c11[i], gxf - gx, gyf - gy).int
        }

        out.setpixel(x => x, y => y, color => rgb)
    }

    return out
}

var img = %O<Imager>.new(file => "input.png")
var out = scale(img, 1.6, 1.6)
out.write(file => "output.png")
```

Binary digits

```
[5, 50, 9000].each { |n|
    say n.as_bin
}
```

Output:

```
101
110010
10001100101000
```

Binary search

Iterative

```
func binary_search(a, i) {

    var l = 0
    var h = a.end

    while (l <= h) {
        var mid = (h+l / 2 -> int)
        a[mid] > i && (h = mid-1; next)
        a[mid] < i && (l = mid+1; next)
        return mid
    }

    return -1
}
```

Recursive

```
func binary_search(arr, value, low=0, high=arr.end) {
    high < low && return -1
    var middle = ((high+low) // 2)

    given (arr[middle]) { |item|
        case (value < item) {
            binary_search(arr, value, low, middle-1)
        }
        case (value > item) {
            binary_search(arr, value, middle+1, high)
        }
        case (value == item) {
            middle
        }
    }
}
```

Usage example:

```
say binary_search([34, 42, 55, 778], 55)       #=> 2
```

Bitmap/Bresenham's line algorithm

```
func my_draw_line(img, x0, y0, x1, y1) {

    var steep = (abs(y1 - y0) > abs(x1 - x0))

    if (steep) {
        (y0, x0) = (x0, y0)
        (y1, x1) = (x1, y1)
    }
    if (x0 > x1) {
        (x1, x0) = (x0, x1)
        (y1, y0) = (y0, y1)
    }

    var deltax = (x1 - x0)
    var deltay = abs(y1 - y0)
    var error  = (deltax / 2)
    var y = y0
    var ystep = (y0 < y1 ? 1 : -1)

    for x (x0 .. x1) {
        img.draw_point(steep ? ((y, x)) : ((x, y)))
        error -= deltay
        if (error < 0) {
            y += ystep
            error += deltax
        }
    }
}

require('Image::Imlib2')

var img = %s'Image::Imlib2'.new(160, 160)
img.set_color(255, 255, 255, 255) # white
img.fill_rectangle(0,0,160,160)

img.set_color(0,0,0,255) # black
my_draw_line(img, 10, 80, 80, 160)
my_draw_line(img, 80, 160, 160, 80)
my_draw_line(img, 160, 80, 80, 10)
my_draw_line(img, 80, 10, 10, 80)

img.save("test0.png")

# let's try the same using its internal algo
img.set_color(255, 255, 255, 255) # white
img.fill_rectangle(0,0,160,160)
img.set_color(0,0,0,255) # black
img.draw_line(10, 80, 80, 160)
img.draw_line(80, 160, 160, 80)
img.draw_line(160, 80, 80, 10)
img.draw_line(80, 10, 10, 80)

img.save("test1.png")
```

Bitmap/Write a PPM file

```
subset Int   < Number {|n| n.is_int  }
subset UInt  < Int    {|n| n >= 0    }
subset UInt8 < Int    {|n| n ~~ ^256 }

struct Pixel {
    R < UInt8,
    G < UInt8,
    B < UInt8
}

class Bitmap(width < UInt, height < UInt) {
    has data = []

    method fill(Pixel p) {
        data = (width*height -> of { Pixel(p.R, p.G, p.B) })
    }

    method setpixel(i < UInt, j < UInt, Pixel p) {

        subset WidthLimit  < UInt { |n| n ~~ ^width  }
        subset HeightLimit < UInt { |n| n ~~ ^height }

        func (w < WidthLimit, h < HeightLimit) {
            data[w*height + h] = p
        }(i, j)
    }

    method p6 {
        <<-EOT + data.map {|p| [p.R, p.G, p.B].pack('C3') }.join
        P6
        #{width} #{height}
        255
        EOT
    }
}

var b = Bitmap(width: 125, height: 125)

for i,j in (^b.height ~X ^b.width) {
    b.setpixel(i, j, Pixel(2*i, 2*j, 255 - 2*i))
}

%f"palette.ppm".write(b.p6, :raw)
```

Bitwise operations

```
func bitwise(a, b) {
   say ('a and b : ',  a & b)
   say ('a or b  : ',  a | b)
   say ('a xor b : ',  a ^ b)
   say ('not a   : ',     ~a)
   say ('a << b  : ', a << b)  # left shift
   say ('a >> b  : ', a >> b)  # arithmetic right shift
}

bitwise(14,3)
```

Output:

```
a and b : 2
a or b  : 15
a xor b : 13
not a   : -15
a << b  : 112
a >> b  : 1
```

Boolean values

Sidef defines the *true* and *false* boolean values, which are part of the *Bool* type.

```
var t = true
var f = false
```

In conditional expressions, anything that evaluates to zero or nothing is considered *false*, including empty arrays and empty hashes.

```
if (0 || "0" || false || nil || "" || [] || :()) {
    say "true"
} else {
    say "false"
}
```

Output:

```
false
```

Box the compass

```
func point (index) {
    var ix = (index % 32)
    if    (ix & 1) { "#{point((ix + 1) & 28)} by #{point(((2 - (ix & 2)) * 4) + ix & 24)}" }
    elsif (ix & 2) { "#{point((ix + 2) & 24)}-#{point((ix | 4) & 28)}" }
    elsif (ix & 4) { "#{point((ix + 8) & 16)}#{point((ix | 8) & 24)}" }
    else           { <north east south west>[ix / 8] }
}

func test_angle (ix) { ix * 11.25 + [0, 5.62, -5.62][ ix % 3 ] }
func angle_to_point(θ) { (θ / 360 * 32) + 0.5 -> floor }

for ix in range(0, 32) {
    var θ = test_angle(ix)
    printf("  %2d %6.2f° %s\n", ix % 32 + 1, θ, point(angle_to_point(θ)).tc)
}
```

Output:

```
 1   0.00° North
 2  16.87° North by east
 3  16.88° North-northeast
 4  33.75° Northeast by north
 5  50.62° Northeast
 6  50.63° Northeast by east
 7  67.50° East-northeast
 8  84.37° East by north
 9  84.38° East
10 101.25° East by south
11 118.12° East-southeast
12 118.13° Southeast by east
13 135.00° Southeast
14 151.87° Southeast by south
15 151.88° South-southeast
16 168.75° South by east
17 185.62° South
18 185.63° South by west
19 202.50° South-southwest
20 219.37° Southwest by south
21 219.38° Southwest
22 236.25° Southwest by west
23 253.12° West-southwest
24 253.13° West by south
25 270.00° West
26 286.87° West by north
27 286.88° West-northwest
28 303.75° Northwest by west
29 320.62° Northwest
30 320.63° Northwest by north
31 337.50° North-northwest
32 354.37° North by west
 1 354.38° North
```

Brace expansion

```
func brace_expand (input) {
    var current = ['']
    var stack = [[current]]

    loop {
        var t = input.match(
            /\G ((?:[^\\{,}]++ | \\(?:.|\z))++ | . )/gx
        )[0] \\ break

        if (t == '{') {
            stack << [current = ['']]
        }
        elsif ((t == ',') && (stack.len > 1)) {
            stack[-1] << (current = [''])
        }
        elsif ((t == '}') && (stack.len > 1)) {
            var group = stack.pop
            current   = stack[-1][-1]

            # handle the case of brace pairs without commas:
            group[0][] = group[0].map{ '{'+_+'}' }... if (group.len == 1)

            current[] = current.map { |c|
                group.map { .map { c + _ }... }...
            }...
        }
        else {
            current[] = current.map { _ + t }...
        }
    }

    # handle the case of missing closing braces:
    while (stack.len > 1) {

        var right = stack[-1].pop
        var sep = ','

        if (stack[-1].is_empty) {
            sep = '{'
            stack.pop
        }

        current = stack[-1][-1]
        current[] = current.map  { |c|
            right.map { c + sep + _ }...
        }...
    }

    return current
}

DATA.each { |line|
    say line
    brace_expand(line).each { "\t#{_}".say }
    say ''
}

__DATA__
~/{Downloads,Pictures}/*.{jpg,gif,png}
It{{em,alic}iz,erat}e{d,}, please.
{,{,gotta have{ ,\, again\, }}more }cowbell!
{}} some }{,{\\{ edge, edge} \,}{ cases, {here} \\\\\}
```

Output:

```
~/{Downloads,Pictures}/*.{jpg,gif,png}
        ~/Downloads/*.jpg
        ~/Downloads/*.gif
        ~/Downloads/*.png
        ~/Pictures/*.jpg
        ~/Pictures/*.gif
        ~/Pictures/*.png

It{{em,alic}iz,erat}e{d,}, please.
        Itemized, please.
        Itemize, please.
        Italicized, please.
        Italicize, please.
        Iterated, please.
        Iterate, please.

{,{,gotta have{ ,\, again\, }}more }cowbell!
        cowbell!
        more cowbell!
        gotta have more cowbell!
        gotta have\, again\, more cowbell!

{}} some }{,{\\{ edge, edge} \,}{ cases, {here} \\\\\}
        {}} some }{,{\\ edge \,}{ cases, {here} \\\\\}
        {}} some }{,{\\ edge \,}{ cases, {here} \\\\\}
```

Brazilian numbers

```
func is_Brazilian_prime(q) {

    static L = Set()
    static M = 0

    return true  if L.has(q)
    return false if (q < M)

    var N = (q<1000 ? 1000 : 2*q)

    for K in (primes(3, ilog2(N+1))) {
        for n in (2 .. iroot(N-1, K-1)) {
            var p = (n**K - 1)/(n-1)
            L << p if (p<N && p.is_prime)
        }
    }

    M = (L.max \\ 0)
    return L.has(q)
}

func is_Brazilian(n) {

    if (!n.is_prime) {
        n.is_square || return (n>6)
        var m = n.isqrt
        return (m>3 && (!m.is_prime || m==11))
    }

    is_Brazilian_prime(n)
}


with (20) {|n|
    say "First #{n} Brazilian numbers:"
    say (^Inf -> lazy.grep(is_Brazilian).first(n))

    say "\nFirst #{n} odd Brazilian numbers:"
    say (^Inf -> lazy.grep(is_Brazilian).grep{.is_odd}.first(n))

    say "\nFirst #{n} prime Brazilian numbers"
    say (^Inf -> lazy.grep(is_Brazilian).grep{.is_prime}.first(n))
}
```

Output:

```
First 20 Brazilian numbers:
[7, 8, 10, 12, 13, 14, 15, 16, 18, 20, 21, 22, 24, 26, 27, 28, 30, 31, 32, 33]

First 20 odd Brazilian numbers:
[7, 13, 15, 21, 27, 31, 33, 35, 39, 43, 45, 51, 55, 57, 63, 65, 69, 73, 75, 77]

First 20 prime Brazilian numbers
[7, 13, 31, 43, 73, 127, 157, 211, 241, 307, 421, 463, 601, 757, 1093, 1123, 1483, 1723, 2551, 2801]
```

Extra:

```
for n in (1..6) {
    say ("#{10**n->commify}th Brazilian number = ", is_Brazilian.nth(10**n))
}
```

Output:

```
10th Brazilian number = 20
100th Brazilian number = 132
1,000th Brazilian number = 1191
10,000th Brazilian number = 11364
100,000th Brazilian number = 110468
1,000,000th Brazilian number = 1084566
```

Break OO privacy

Sidef's object model does not enforce privacy, but it allows storing private attributes inside the container of an object, which is an hash:

```
class Example {
    has public = "foo"
    method init {
        self{:private} = "secret"
    }
}

var obj = Example();

# Access public attributes
say obj.public;                 #=> "foo"
say obj{:public};               #=> "foo"

# Access private attributes
say obj{:private};              #=> "secret"
```

Brilliant numbers

```
func is_briliant_number(n) {
    n.is_semiprime && (n.factor.map{.len}.uniq.len == 1)
}

func brilliant_numbers_count(n) {

    var count = 0
    var len = n.isqrt.len

    for k in (1 .. len-1) {
        var pi = prime_count(10**(k-1), 10**k - 1)
        count += binomial(pi, 2)+pi
    }

    var min = (10**(len - 1))
    var max = (10**len - 1)

    each_prime(min, max, {|p|
        count += prime_count(p, max `min` idiv(n, p))
    })

    return count
}

say "First 100 brilliant numbers:"

100.by(is_briliant_number).each_slice(10, {|*a|
    say a.map { '%4s' % _}.join(' ')
})

say ''

for n in (1 .. 12) {
    var v = (10**n .. Inf -> first_by(is_briliant_number))
    printf("First brilliant number >= 10^%d is %s", n, v)
    printf(" at position %s\n", brilliant_numbers_count(v))
}
```

Output:

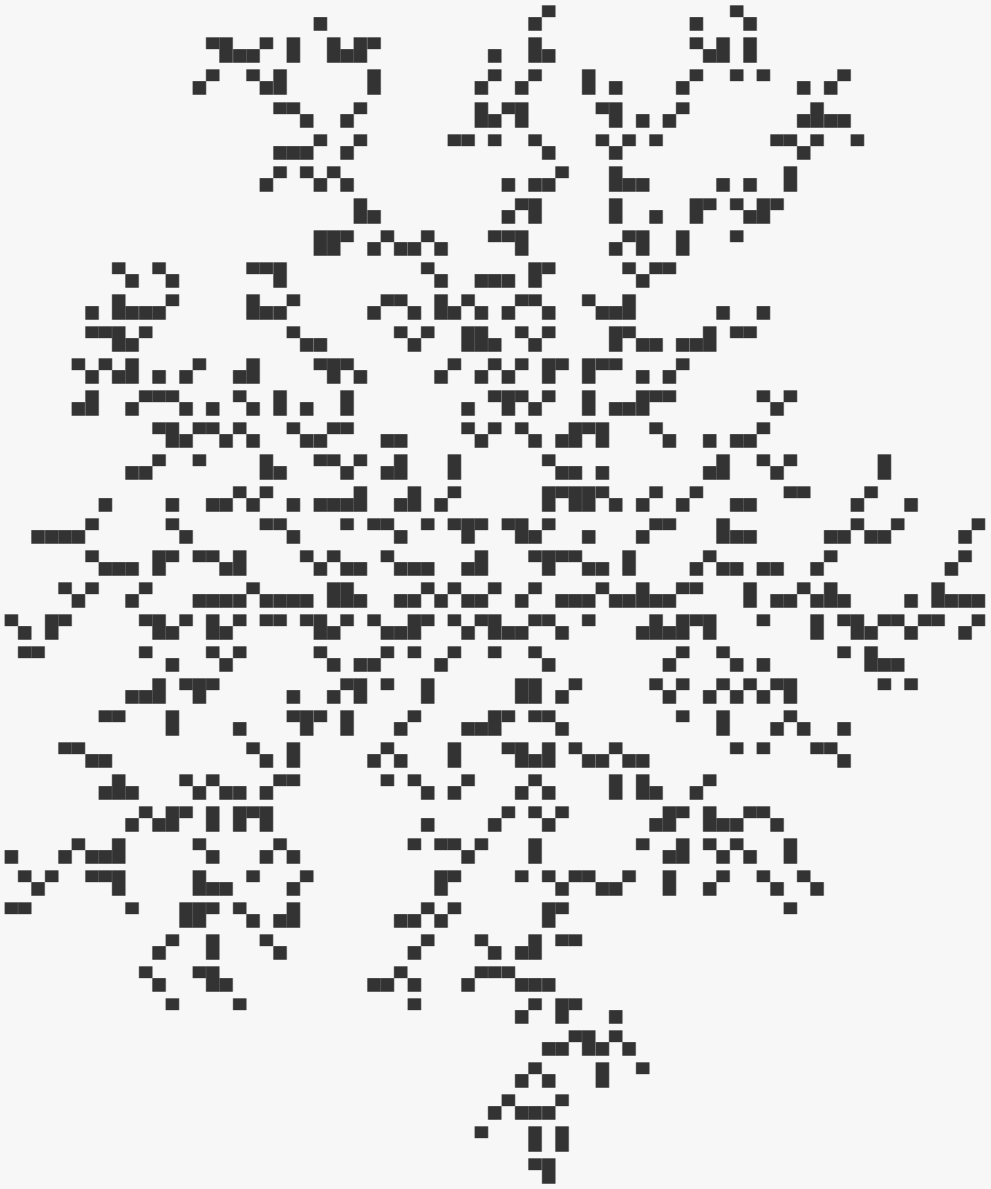
```
First 100 brilliant numbers:
   4    6    9   10   14   15   21   25   35   49
 121  143  169  187  209  221  247  253  289  299
 319  323  341  361  377  391  403  407  437  451
 473  481  493  517  527  529  533  551  559  583
 589  611  629  649  667  671  689  697  703  713
 731  737  767  779  781  793  799  803  817  841
 851  869  871  893  899  901  913  923  943  949
 961  979  989 1003 1007 1027 1037 1067 1073 1079
1081 1121 1139 1147 1157 1159 1189 1207 1219 1241
1247 1261 1271 1273 1333 1343 1349 1357 1363 1369

First brilliant number >= 10^1 is 10 at position 4
First brilliant number >= 10^2 is 121 at position 11
First brilliant number >= 10^3 is 1003 at position 74
First brilliant number >= 10^4 is 10201 at position 242
First brilliant number >= 10^5 is 100013 at position 2505
First brilliant number >= 10^6 is 1018081 at position 10538
First brilliant number >= 10^7 is 10000043 at position 124364
First brilliant number >= 10^8 is 100140049 at position 573929
First brilliant number >= 10^9 is 1000000081 at position 7407841
First brilliant number >= 10^10 is 10000600009 at position 35547995
First brilliant number >= 10^11 is 100000000147 at position 491316167
First brilliant number >= 10^12 is 1000006000009 at position 2409600866
```

Brownian tree

```
const size = 100
const mid = size>>1
const particlenum = 1000

var map = []
var spawnradius = 5

func set(x, y) {
    map[x][y] = 1
}

func get(x, y) {
    map[x][y] \\ 0
}

set(mid, mid)

var blocks = [
    " ",
    "\N{UPPER HALF BLOCK}",
    "\N{LOWER HALF BLOCK}",
    "\N{FULL BLOCK}"
]

func block(a, b) {
    blocks[2*b + a]
}

func display {
    0..size `by` 2 -> map {|y|
        0..size -> map {|x|
            if ([x, y].all { .-mid < spawnradius }) {
                block(get(x, y), get(x, y+1))
            } else { " " }
```

```
        }.join
    }.join("\n").say
}

for progress in (^particlenum) {
    var (x=0, y=0)

    var reset = {
        do {
            (x, y) = (
                (mid-spawnradius .. mid+spawnradius  -> pick),
                [mid-spawnradius,   mid+spawnradius] -> pick
            )
            (x, y) = (y, x) if (1.rand < 0.5)
        } while(get(x, y))
    }

    reset.run

    while ([[-1, 0, 1]]*2 -> cartesian.any {|pair|
        get(x+pair[0], y+pair[1])
    } -> not) {
        x = [x-1, x, x+1].pick
        y = [y-1, y, y+1].pick

        if (1.rand < 0.25) {
            x = (x >= mid ? (x-1) : (x+1))
            y = (y >= mid ? (y-1) : (y+1))
        }

        if ([x,y].any { .-mid > spawnradius }) {
            reset.run
        }
    }

    set(x, y)
    display() if (progress %% 50)

    if ((spawnradius < mid) && [x,y].any { .-mid > spawnradius-5 }) {
        ++spawnradius
    }
}

display()
```

Output:



Bulls and cows

```
var size = 4
var num = @(1..9).shuffle.first(size)

for (var guesses = 1; true; guesses++) {

    var bulls = 0
    var cows  = 0

    var input =
        read("Input: ", String).chars           \
                               .uniq            \
                               .grep(/^[1-9]$/) \
                               .map{.to_n}

    if (input.len != size) {
        warn "Invalid input!\n"
        guesses--
        next
    }

    if (input == num) {
        printf("You did it in %d attempts!\n", guesses)
        break
    }

    for i (^num) {
        if (num[i] == input[i]) {
            bulls++
        }
        elsif (num.contains(input[i])) {
            cows++
        }
    }

    "Bulls: %d; Cows: %d\n".printf(bulls, cows)
}
```

Output:

```
Input: 2953
Bulls: 1; Cows: 1
Input: 9654
Bulls: 1; Cows: 1
Input: 8924
Bulls: 1; Cows: 3
Input: 2894
You did it in 4 attempts!
```

Bulls and cows/Player

```
# Build a list of all possible solutions.  The regular expression weeds
# out numbers containing zeroes or repeated digits.
var candidates = (1234..9876 -> grep {|n| !("#{n}" =~ /0 | (\d) .*? \1 /x) }.map{.digits});

# Repeatedly prompt for input until the user supplies a reasonable score.
# The regex validates the user's input and then returns two numbers.
func read_score(guess) {
    loop {
        "My guess: %s   (from %d possibilities)\n" \
            -> printf(guess.join, candidates.len);

        if (var m = (Sys.scanln("bulls cows: ") =~ /^\h*(\d)\h*(\d)\h*$/)) {
            var (bulls, cows) = m.cap.map{.to_i}...;
            bulls+cows <= 4 && return(bulls, cows);
        }

        say "Please specify the number of bulls and the number of cows";
    }
}

func score_correct(a, b, bulls, cows) {
    var (exact, loose) = (0, 0);

    for i in ^4 {
        a[i] == b[i] ? ++exact
                     : (a[i]~~b && ++loose)
    }

    (bulls == exact) && (cows == loose)
}

# Pick a number, display it, get the score, and discard candidates
# that don't match the score:
loop {
    var guess = candidates.pick;
    var (bulls, cows) = read_score(guess);
    candidates.grep!{|n| score_correct(n, guess, bulls, cows) }
    candidates.len > 1 || break
}

# Print the secret number or the error message
say (
    candidates.len == 1 ? ("Your secret number is: %d" % candidates[0].join)
                        : ("I think you made a mistake with your scoring")
)
```

Output:

Output:

```
My guess: 7432   (from 3024 possibilities)
bulls cows: 0 1
My guess: 9216   (from 720 possibilities)
bulls cows: 1 1
My guess: 6813   (from 128 possibilities)
bulls cows: 0 1
My guess: 9157   (from 24 possibilities)
bulls cows: 1 3
Your secret number is: 9571
```

Burrows–Wheeler transform

```
class BurrowsWheelerTransform (String L = "\002") {

    method encode(String s) {
        assert(!s.contains(L), "String cannot contain `#{L.dump}`")
        s = (L + s)
        s.len.of{|i| s.substr(i) + s.substr(0, i) }.sort.map{.last}.join
    }

    method decode(String s) {
        var t = s.len.of("")
        var c = s.chars
        { t = (c »+« t).sort } * s.len
        t.first { .begins_with(L) }.substr(L.len)
    }
}

var tests = [
    "banana", "appellee", "dogwood", "TOBEORNOTTOBEORTOBEORNOT"
    "SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES",
]

var bwt = BurrowsWheelerTransform(L: '$')

tests.each { |str|
    var enc = bwt.encode(str)
    var dec = bwt.decode(enc)
    say "BWT(#{dec.dump}) = #{enc.dump}"
    assert_eq(str, dec)
}
```

Output:

```
BWT("banana") = "annb$aa"
BWT("appellee") = "e$elplepa"
BWT("dogwood") = "do$oodwg"
BWT("TOBEORNOTTOBEORTOBEORNOT") = "TOOOBBBRRTTTEEENNOOOOR$TO"
BWT("SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES") = "STEXYDST.E.IXXIIXXSSMPPS.B..EE.$.USFXDIIOIIIT"
```

Caesar cipher

```
func caesar(msg, key, decode=false) {
    decode && (key = (26 - key))
    msg.gsub(/([A-Z])/i, {|c| ((c.uc.ord - 65 + key) % 26) + 65 -> chr})
}

var msg = 'THE FIVE BOXING WIZARDS JUMP QUICKLY'

var enc = caesar(msg, 10)
var dec = caesar(enc, 10, true)

say "msg: #{msg}"
say "enc: #{enc}"
say "dec: #{dec}"
```

Output:

```
msg: THE FIVE BOXING WIZARDS JUMP QUICKLY
enc: DRO PSFO LYHSXQ GSJKBNC TEWZ AESMUVI
dec: THE FIVE BOXING WIZARDS JUMP QUICKLY
```

Calculating the value of e

```
func calculate_e(n=50) {
    sum(0..n, {|k| 1/k! })
}

say calculate_e()
say calculate_e(69).as_dec(100)
```

Output:

```
2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642 7
2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642 7
```

For finding the number of required terms for calculating `e` to a given number of decimal places, using the formula `Sum_{k=0..n} 1/k!`, we have:

```
func f(n) {
    var t = n*log(10)
    (n + 10).bsearch_le { |k|
        lngamma(k+1) <=> t
    }
}

for k in (1..10) {
    var n = f(10**k)
    say "Sum_{k=0..#{n}} 1/k! = e correct to #{10**k->commify} decimal places"
}
```

Output:

```
Sum_{k=0..13} 1/k! = e correct to 10 decimal places
Sum_{k=0..69} 1/k! = e correct to 100 decimal places
Sum_{k=0..449} 1/k! = e correct to 1,000 decimal places
Sum_{k=0..3248} 1/k! = e correct to 10,000 decimal places
Sum_{k=0..25205} 1/k! = e correct to 100,000 decimal places
Sum_{k=0..205022} 1/k! = e correct to 1,000,000 decimal places
Sum_{k=0..1723507} 1/k! = e correct to 10,000,000 decimal places
Sum_{k=0..14842906} 1/k! = e correct to 100,000,000 decimal places
Sum_{k=0..130202808} 1/k! = e correct to 1,000,000,000 decimal places
Sum_{k=0..1158787577} 1/k! = e correct to 10,000,000,000 decimal places
```

Calendar

```
require('DateTime')

define months_per_col = 3
define week_day_names = <Mo Tu We Th Fr Sa Su>
define month_names = <Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec>

func fmt_month (year, month) {
    var str = sprintf("%-20s\n", month_names[month-1])
    str += week_day_names.join(' ')+"\n"

    var dt = %O<DateTime>
    var date = dt.new(year => year, month => month)
    var week_day = date.day_of_week
    str += (week_day-1 `of` "  " -> join(" "))

    var last_day = dt.last_day_of_month(year => year, month => month).day
    for day (date.day .. last_day) {
        date = dt.new(year => year, month => month, day => day)
        str += " " if (week_day ~~ (2..7))
        if (week_day == 8) {
            str += "\n"
            week_day = 1
        }
        str += sprintf("%2d", day)
        ++week_day
    }
    str += " " if (week_day < 8)
    str += (8-week_day `of` "  " -> join(" "))
    str += "\n"
}

func fmt_year (year) {
    var month_strs = 12.of {|i| fmt_month(year, i+1).lines }

    var str = (' '*30 + year + "\n")
    for month (0..11 `by` months_per_col) {
        while (month_strs[month]) {
            for i (1..months_per_col) {
                month_strs[month + i - 1] || next
                str += month_strs[month + i - 1].shift
                str += ' '*3
            }
            str += "\n"
        }
        str += "\n"
    }

    return str
}

print fmt_year(ARGV ? Number(ARGV[0]) : 1969)
```

Output:

```
                                1969
Jan                    Feb                    Mar
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2                   1  2
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30

Apr                    May                    Jun
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6             1  2  3  4                      1
 7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
14 15 16 17 18 19 20   12 13 14 15 16 17 18    9 10 11 12 13 14 15
21 22 23 24 25 26 27   19 20 21 22 23 24 25   16 17 18 19 20 21 22
28 29 30               26 27 28 29 30 31      23 24 25 26 27 28 29

Jul                    Aug                    Sep
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                1  2  3    1  2  3  4  5  6  7
 7  8  9 10 11 12 13    4  5  6  7  8  9 10    8  9 10 11 12 13 14
14 15 16 17 18 19 20   11 12 13 14 15 16 17   15 16 17 18 19 20 21
21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
28 29 30 31            25 26 27 28 29 30 31   29 30

Oct                    Nov                    Dec
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2    1  2  3  4  5  6  7
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
13 14 15 16 17 18 19   10 11 12 13 14 15 16   15 16 17 18 19 20 21
20 21 22 23 24 25 26   17 18 19 20 21 22 23   22 23 24 25 26 27 28
27 28 29 30 31         24 25 26 27 28 29 30   29 30 31
```

Calendar - for "REAL" programmers

```
-> DT { ('DATE'.("\LWC") + 'TIME'.("\LWC")).("\LREQUIRE") }

-> MONTHS_PER_COL { 6 }
-> WEEK_DAY_NAMES { <MO TU WE TH FR SA SU> }
-> MONTH_NAMES    { <JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC> }

-> FMT_MONTH (YEAR, MONTH, STR="", WEEK_DAY=0) {
    STR  = "%11\LS\E%9\LS\E\12".("\LSPRINTF")(MONTH_NAMES()[MONTH-1],'')
    STR += (WEEK_DAY_NAMES().("\LJOIN")(' ') + "\12")

    -> DATE { DT().("\LNEW")("\LYEAR" => YEAR, "\LMONTH" => MONTH) }

    WEEK_DAY = DATE().("\LDAY_OF_WEEK")
    STR += (["  "] * WEEK_DAY-1 -> ("\LJOIN")(" "))

    -> LAST_DAY {
        DT().("\LLAST_DAY_OF_MONTH")(
            "\LYEAR" => YEAR, "\LMONTH" => MONTH
        ).("\LDAY")
    }

    (DATE().("\LDAY") .. LAST_DAY()).("\LEACH")({ |DAY|
        (WEEK_DAY ~~ (2..7)) && (STR += " ")

        (WEEK_DAY == 8) && (
            STR += "\12"
            WEEK_DAY = 1
        )
        STR += ("%2\LD" % DAY)
        ++WEEK_DAY
    })
    (WEEK_DAY < 8) && (STR += " ")
    STR += (["  "] * 8-WEEK_DAY -> ("\LJOIN")(" "))
    STR += "\12"
}

-> FMT_YEAR (YEAR, STR="", MONTH_STRS=[]) {
    MONTH_STRS = 12.("\LOF")({|I| FMT_MONTH(YEAR, I+1).("\LLINES") })

    STR += (' '*(MONTHS_PER_COL()*10 + 2) + YEAR + "\12")
    (0..11 -> ("\LBY")(MONTHS_PER_COL())).("\LEACH")({ |MONTH|
        MONTH_STRS[MONTH] && ->() {
            { |I|
                MONTH_STRS[MONTH + I] && (
                    STR += MONTH_STRS[MONTH + I].("\LSHIFT")
                    STR += ' '*2
                )
            } * MONTHS_PER_COL()

            STR += "\12"
            MONTH_STRS[MONTH] && __FUNC__()
        }()
        STR += "\12"
    })

    STR
}

FMT_YEAR(ARGV ? ARGV[0].("\LTO_I") : 1969).("\LPRINT")
```

Output:

```
                                                1969
        JAN                     FEB                     MAR                     APR                     MAY
MO TU WE TH FR SA SU  MO TU WE TH FR SA SU  MO TU WE TH FR SA SU  MO TU WE TH FR SA SU  MO TU WE TH FR
       1  2  3  4  5                  1  2                  1  2      1  2  3  4  5  6           1  2
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   3  4  5  6  7  8  9   7  8  9 10 11 12 13   5  6  7  8  9
13 14 15 16 17 18 19  10 11 12 13 14 15 16  10 11 12 13 14 15 16  14 15 16 17 18 19 20  12 13 14 15 16
20 21 22 23 24 25 26  17 18 19 20 21 22 23  17 18 19 20 21 22 23  21 22 23 24 25 26 27  19 20 21 22 23
27 28 29 30 31        24 25 26 27 28        24 25 26 27 28 29 30  28 29 30              26 27 28 29 30

        JUL                     AUG                     SEP                     OCT                     NOV
MO TU WE TH FR SA SU  MO TU WE TH FR SA SU  MO TU WE TH FR SA SU  MO TU WE TH FR SA SU  MO TU WE TH FR
    1  2  3  4  5  6               1  2  3   1  2  3  4  5  6  7        1  2  3  4  5
 7  8  9 10 11 12 13   4  5  6  7  8  9 10   8  9 10 11 12 13 14   6  7  8  9 10 11 12   3  4  5  6  7
14 15 16 17 18 19 20  11 12 13 14 15 16 17  15 16 17 18 19 20 21  13 14 15 16 17 18 19  10 11 12 13 14
21 22 23 24 25 26 27  18 19 20 21 22 23 24  22 23 24 25 26 27 28  20 21 22 23 24 25 26  17 18 19 20 21
28 29 30 31           25 26 27 28 29 30 31  29 30                 27 28 29 30 31        24 25 26 27 28
```

Calkin-Wilf sequence

```
func calkin_wilf(n) is cached {
    return 1 if (n == 1)
    1/(2*floor(__FUNC__(n-1)) + 1 - __FUNC__(n-1))
}

func r2cw(r) {

    var cfrac = r.as_cfrac
    cfrac.len.is_odd || return nil

    Num(cfrac.flip.map_kv {|k,v| (k.is_odd ? '0' : '1') * v }.join, 2)
}

with (20) {|n|
    say "First #{n} terms of the Calkin-Wilf sequence:"
    say calkin_wilf.map(1..n)
}

with (83116/51639) {|r|
    say ("\n#{r.as_rat} is at index: ", r2cw(r))
}
```

Output:

```
First 20 terms of the Calkin-Wilf sequence:
[1, 1/2, 2, 1/3, 3/2, 2/3, 3, 1/4, 4/3, 3/5, 5/2, 2/5, 5/3, 3/4, 4, 1/5, 5/4, 4/7, 7/3, 3/8]

83116/51639 is at index: 123456789
```

Call a function

All functions in Sidef are first-class closures

```
foo()                       # without arguments
foo(1, 2)                   # with two arguments
foo(args...)                # with a variable number of arguments
foo(name: 'Bar', age: 42)   # with named arguments

var f = foo                 # store the function foo in variable 'f'
var result = f()            # obtain the return value of a function

var arr = [1,2,3]
foo(arr)                    # the arguments are passed by object-reference
```

Partial application is possible by using a curry function:

```
func curry(f, *args1) {
    func (*args2) {
        f(args1..., args2...)
    }
}

func add(a, b) {
    a + b
}

var adder = curry(add, 1)
say adder(3)                 #=>4
```

Call an object method

```
class MyClass {
    method foo(arg) { say arg }
}

var arg = 42

# Call a class method
MyClass.foo(arg)

# Alternatively, using an expression for the method name
MyClass.(:foo)(arg)

# Create an instance
var instance = MyClass()

# Instance method
instance.foo(arg)

# Alternatively, by using an expression for the method name
instance.(:foo)(arg)

# Alternatively, by asking for a method
instance.method(:foo)(arg)
```

Cantor set

```
func cantor (height) {
    var width = 3**(height - 1)
    var lines = height.of { "\N{FULL BLOCK}" * width }

    func trim_middle_third (len, start, index) {
        var seg = (len // 3) || return()

        for i, j in ((index ..^ height) ~X (0 ..^ seg)) {
            lines[i].replace!(Regex("^.{#{start + seg + j}}\\K."), ' ')
        }

        [0, 2*seg].each { |k|
            trim_middle_third(seg, start + k, index + 1)
        }
    }

    trim_middle_third(width, 0, 1)
    return lines
}

cantor(5).each { .say }
```

Output:

Carmichael 3 strong pseudoprimes

```
func forprimes(a, b, callback) {
    for (a = (a-1 -> next_prime); a <= b; a.next_prime!) {
        callback(a)
    }
}

forprimes(3, 61, func(p) {
   for h3 in (2 ..^ p) {
      var ph3 = (p + h3)
      for d in (1 ..^ ph3) {
         ((-p * p) % h3) != (d % h3) && next
         ((p-1) * ph3) % d && next
         var q = 1+((p-1) * ph3 / d)
         q.is_prime || next
         var r = 1+((p*q - 1)/h3)
         r.is_prime || next
         (q*r) % (p-1) == 1 || next
         printf("%2d x %5d x %5d = %s\n",p,q,r, p*q*r)
      }
   }
})
```

Output:

```
 3 x    11 x    17 = 561
 5 x    29 x    73 = 10585
 5 x    17 x    29 = 2465
 5 x    13 x    17 = 1105
 ... full output is 69 lines ...
61 x   661 x  2521 = 101649241
61 x   271 x   571 = 9439201
61 x   241 x   421 = 6189121
61 x  3361 x  4021 = 824389441
```

Cartesian product of two or more lists

In Sidef, the Cartesian product of an arbitrary number of arrays is built-in as *Array.cartesian()*:

```
cartesian([[1,2], [3,4], [5,6]]).say
cartesian([[1,2], [3,4], [5,6]], {|*arr| say arr })
```

Alternatively, a simple recursive implementation:

```
func cartesian_product(*arr) {

    var c = []
    var r = []

    func {
        if (c.len < arr.len) {
            for item in (arr[c.len]) {
                c.push(item)
                __FUNC__()
                c.pop
            }
        }
        else {
            r.push([c...])
        }
    }()

    return r
}
```

Completing the task:

```
say cartesian_product([1,2], [3,4])
say cartesian_product([3,4], [1,2])
```

Output:

```
[[1, 3], [1, 4], [2, 3], [2, 4]]
[[3, 1], [3, 2], [4, 1], [4, 2]]
```

The product of an empty list with any other list is empty:

```
say cartesian_product([1,2], [])
say cartesian_product([], [1,2])
```

Output:

```
[]
[]
```

Extra credit:

```
cartesian_product([1776, 1789], [7, 12], [4, 14, 23], [0, 1]).each{ .say }
```

Output:

```
[1776, 7, 4, 0]
[1776, 7, 4, 1]
[1776, 7, 14, 0]
[1776, 7, 14, 1]
[1776, 7, 23, 0]
[1776, 7, 23, 1]
[1776, 12, 4, 0]
[1776, 12, 4, 1]
[1776, 12, 14, 0]
[1776, 12, 14, 1]
[1776, 12, 23, 0]
[1776, 12, 23, 1]
[1789, 7, 4, 0]
[1789, 7, 4, 1]
[1789, 7, 14, 0]
[1789, 7, 14, 1]
[1789, 7, 23, 0]
[1789, 7, 23, 1]
[1789, 12, 4, 0]
[1789, 12, 4, 1]
[1789, 12, 14, 0]
[1789, 12, 14, 1]
[1789, 12, 23, 0]
[1789, 12, 23, 1]
```

```
say cartesian_product([1, 2, 3], [30], [500, 100])
say cartesian_product([1, 2, 3], [], [500, 100])
```

Output:

```
[[1, 30, 500], [1, 30, 100], [2, 30, 500], [2, 30, 100], [3, 30, 500], [3, 30, 100]]
[]
```

Case-sensitivity of identifiers

```
var dog = 'Benjamin'
var Dog = 'Samba'
var DOG = 'Bernie'
say "The three dogs are named #{dog}, #{Dog}, and #{DOG}."
```

Output:

```
The three dogs are named Benjamin, Samba, and Bernie.
```

Casting out nines

```
func cast_out(base = 10, min = 1, max = (base**2 - 1)) {

    var b9  = base-1
    var ran = b9.range.grep {|n| n%b9 == (n*n % b9) }

    var x = min//b9
    var r = []

    loop {
        ran.each {|n|
            var k = (b9*x + n)
            return r if (k > max)
            r << k if (k >= min)
        }
        ++x
    }

    return r
}

say cast_out().join(' ')
say cast_out(16).join(' ')
say cast_out(17).join(' ')
```

Output:

```
1 9 10 18 19 27 28 36 37 45 46 54 55 63 64 72 73 81 82 90 91 99
1 6 10 15 16 21 25 30 31 36 40 45 46 51 55 60 61 66 70 75 76 81 85 90 91 96 100 105 106 111 115 120 121
1 16 17 32 33 48 49 64 65 80 81 96 97 112 113 128 129 144 145 160 161 176 177 192 193 208 209 224 225 2
```

Catalan numbers

```
func f(i) { i==0 ? 1 : (i * f(i-1)) }
func c(n) { f(2*n) / f(n) / f(n+1) }
```

With memoization:

```
func c(n) is cached {
    n == 0 ? 1 : (c(n-1) * (4 * n - 2) / (n + 1))
}
```

Calling the function:

```
15.times { |i|
    say "#{i}\t#{c(i)}"
}
```

Output:

```
0       1
1       1
2       2
3       5
4       14
5       42
6       132
7       429
8       1430
9       4862
10      16796
11      58786
12      208012
13      742900
14      2674440
```

Catalan numbers/Pascal's triangle

```
func catalan(num) {
  var t = [0, 1]
  (1..num).map { |i|
    flip(^i    ).each {|j| t[j+1] += t[j] }
    t[i+1] = t[i]
    flip(^i.inc).each {|j| t[j+1] += t[j] }
    t[i+1] - t[i]
  }
}

say catalan(15).join(' ')
```

Output:

```
1 2 5 14 42 132 429 1430 4862 16796 58786 208012 742900 2674440 9694845
```

Catamorphism

```
say (1..10 -> reduce('+'))
say (1..10 -> reduce{|a,b| a + b})
```

Change e letters to i in words

```
var file = File("unixdict.txt")

if (!file.exists) {
    require('LWP::Simple')
    say ":: Retrieving #{file} from internet..."
    %S<LWP::Simple>.mirror(
        'https://web.archive.org/web/20180611003215if_/' +
        'http://www.puzzlers.org:80/pub/wordlists/unixdict.txt',
    'unixdict.txt')
}

var words = file.read.words
var dict  = Hash().set_keys(words...)
var count = 0

words.each {|word|

    word.len > 5 || next
    word.contains('e') || next

    var changed = word.gsub('e', 'i')

    if (dict.contains(changed)) {
        printf("%2d: %20s <-> %s\n", ++count, word, changed)
    }
}
```

Output:

```
 1:             analyses <-> analysis
 2:             atlantes <-> atlantis
 3:               bellow <-> billow
 4:               breton <-> briton
 5:               clench <-> clinch
 6:              convect <-> convict
 7:               crises <-> crisis
 8:            diagnoses <-> diagnosis
 9:               enfant <-> infant
10:              enquiry <-> inquiry
11:              frances <-> francis
12:              galatea <-> galatia
13:               harden <-> hardin
14:              heckman <-> hickman
15:             inequity <-> iniquity
16:              inflect <-> inflict
17:             jacobean <-> jacobian
18:               marten <-> martin
19:               module <-> moduli
20:              pegging <-> pigging
21:            psychoses <-> psychosis
22:               rabbet <-> rabbit
23:             sterling <-> stirling
24:             synopses <-> synopsis
25:               vector <-> victor
26:               welles <-> willis
```

Changeable words

```
var file = File("unixdict.txt")

if (!file.exists) {
    require('LWP::Simple')
    say ":: Retrieving #{file} from internet..."
    %S<LWP::Simple>.mirror(
        'https://web.archive.org/web/20180611003215if_/' +
        'http://www.puzzlers.org:80/pub/wordlists/unixdict.txt',
    'unixdict.txt')
}

var words  = file.read.words
var bucket = Hash()
var count = 0

words.each {|word|

    var len = word.len

    len > 11 || next
    bucket{len} := []

    bucket{len}.each{|prev|
        if (prev ^ word -> count("\0") == len-1) {
            printf("%2d: %20s <-> %s\n", ++count, prev, word)
        }
    }
    bucket{len} << word
}
```

Output:

```
 1:         aristotelean <-> aristotelian
 2:       claustrophobia <-> claustrophobic
 3:         committeeman <-> committeemen
 4:       committeewoman <-> committeewomen
 5:        complementary <-> complimentary
 6:         confirmation <-> conformation
 7:        congresswoman <-> congresswomen
 8:         councilwoman <-> councilwomen
 9:         craftsperson <-> draftsperson
10:         eavesdropped <-> eavesdropper
11:         frontiersman <-> frontiersmen
12:       handicraftsman <-> handicraftsmen
13:         incommutable <-> incomputable
14:         installation <-> instillation
15:         kaleidescope <-> kaleidoscope
16:         neuroanatomy <-> neuroanotomy
17:         newspaperman <-> newspapermen
18:         nonagenarian <-> nonogenarian
19:         onomatopoeia <-> onomatopoeic
20:         philanthrope <-> philanthropy
21:         prescription <-> proscription
22:        schizophrenia <-> schizophrenic
23:        shakespearean <-> shakespearian
24:         spectroscope <-> spectroscopy
25:        underclassman <-> underclassmen
26:        upperclassman <-> upperclassmen
```

Chaos game

```
require('Imager')

var width  = 600
var height = 600

var points = [
    [width//2,        0],
    [       0, height-1],
    [height-1, height-1],
]

var img = %O|Imager|.new(
                     xsize => width,
                     ysize => height,
                    )

var color = %O|Imager::Color|.new('#ff0000')
var r = [(width-1).irand, (height-1).irand]

30000.times {
    var p = points.rand

    r[] = (
        (p[0] + r[0]) // 2,
        (p[1] + r[1]) // 2,
    )

    img.setpixel(
        x     => r[0],
        y     => r[1],
        color => color,
    )
}

img.write(file => 'chaos_game.png')
```

Output image

Character codes

```
say 'a'.ord     # => 97
say 97.chr      # => 'a'
```

Chebyshev coefficients

```
func chebft (callback, a, b, n) {

    var bma = (0.5 * b-a)
    var bpa = (0.5 * b+a)

    var pi_n = ((^n »+» 0.5) »*» (Num.pi / n))
    var f = (pi_n »cos()» »*» bma »+» bpa «call« callback)
    var sums = (^n «run« {|i| f »*« ((pi_n »*» i) »cos()») «+» })

    sums »*» (2/n)
}

for v in (chebft(func(v){v.cos}, 0, 1, 10)) {
    say ("%+.10e" % v)
}
```

Output:

```
+1.6471694754e+00
-2.3229937162e-01
-5.3715114622e-02
+2.4582352670e-03
+2.8211905743e-04
-7.7222291558e-06
-5.8985564522e-07
+1.1521427333e-08
+6.5963000351e-10
-1.0022591709e-11
```

Check Machin-like formulas

```
var equationtext = <<'EOT'
  pi/4 = arctan(1/2) + arctan(1/3)
  pi/4 = 2*arctan(1/3) + arctan(1/7)
  pi/4 = 4*arctan(1/5) - arctan(1/239)
  pi/4 = 5*arctan(1/7) + 2*arctan(3/79)
  pi/4 = 5*arctan(29/278) + 7*arctan(3/79)
  pi/4 = arctan(1/2) + arctan(1/5) + arctan(1/8)
  pi/4 = 4*arctan(1/5) - arctan(1/70) + arctan(1/99)
  pi/4 = 5*arctan(1/7) + 4*arctan(1/53) + 2*arctan(1/4443)
  pi/4 = 6*arctan(1/8) + 2*arctan(1/57) + arctan(1/239)
  pi/4 = 8*arctan(1/10) - arctan(1/239) - 4*arctan(1/515)
  pi/4 = 12*arctan(1/18) + 8*arctan(1/57) - 5*arctan(1/239)
  pi/4 = 16*arctan(1/21) + 3*arctan(1/239) + 4*arctan(3/1042)
  pi/4 = 22*arctan(1/28) + 2*arctan(1/443) - 5*arctan(1/1393) - 10*arctan(1/11018)
  pi/4 = 22*arctan(1/38) + 17*arctan(7/601) + 10*arctan(7/8149)
  pi/4 = 44*arctan(1/57) + 7*arctan(1/239) - 12*arctan(1/682) + 24*arctan(1/12943)
  pi/4 = 88*arctan(1/172) + 51*arctan(1/239) + 32*arctan(1/682) + 44*arctan(1/5357) +
68*arctan(1/12943)
  pi/4 = 88*arctan(1/172) + 51*arctan(1/239) + 32*arctan(1/682) + 44*arctan(1/5357) +
68*arctan(1/12944)
EOT

func parse_eqn(equation) {
    static eqn_re = %r{
    (^ \s* pi/4 \s* = \s* )?                # LHS of equation
    (?:                                     # RHS
        \s* ( [-+] )? \s*
        (?: ( \d+ ) \s* \*)?
```

```
        \s* arctan\((.*?)\)
    )}x

    gather {
        for lhs,sign,mult,rat in (equation.findall(eqn_re)) {
            take([
                [+1, -1][sign == '-'] * (mult ? Num(mult) : 1),
                Num(rat)
            ])
        }
    }
}

func tanEval(coef, f) {
    return f if (coef == 1)
    return -tanEval(-coef, f) if (coef < 0)
    var ca = coef>>1
    var cb = (coef - ca)
    var (a, b) = (tanEval(ca, f), tanEval(cb, f))
    (a + b) / (1 - a*b)
}

func tans(xs) {
    var xslen = xs.len
    return tanEval(xs[0]...) if (xslen == 1)
    var (aa, bb) = xs.part(xslen>>1)
    var (a, b) = (tans(aa), tans(bb))
    (a + b) / (1 - a*b)
}

var machins = equationtext.lines.map(parse_eqn)

for machin,eqn in (machins ~Z equationtext.lines) {
    var ans = tans(machin)
    printf("%5s: %s\n", (ans == 1 ? 'OK' : 'ERROR'), eqn)
}
```

Output:

```
   OK:   pi/4 = arctan(1/2) + arctan(1/3)
   OK:   pi/4 = 2*arctan(1/3) + arctan(1/7)
   OK:   pi/4 = 4*arctan(1/5) - arctan(1/239)
   OK:   pi/4 = 5*arctan(1/7) + 2*arctan(3/79)
   OK:   pi/4 = 5*arctan(29/278) + 7*arctan(3/79)
   OK:   pi/4 = arctan(1/2) + arctan(1/5) + arctan(1/8)
   OK:   pi/4 = 4*arctan(1/5) - arctan(1/70) + arctan(1/99)
   OK:   pi/4 = 5*arctan(1/7) + 4*arctan(1/53) + 2*arctan(1/4443)
   OK:   pi/4 = 6*arctan(1/8) + 2*arctan(1/57) + arctan(1/239)
   OK:   pi/4 = 8*arctan(1/10) - arctan(1/239) - 4*arctan(1/515)
   OK:   pi/4 = 12*arctan(1/18) + 8*arctan(1/57) - 5*arctan(1/239)
   OK:   pi/4 = 16*arctan(1/21) + 3*arctan(1/239) + 4*arctan(3/1042)
   OK:   pi/4 = 22*arctan(1/28) + 2*arctan(1/443) - 5*arctan(1/1393) - 10*arctan(1/11018)
   OK:   pi/4 = 22*arctan(1/38) + 17*arctan(7/601) + 10*arctan(7/8149)
   OK:   pi/4 = 44*arctan(1/57) + 7*arctan(1/239) - 12*arctan(1/682) + 24*arctan(1/12943)
   OK:   pi/4 = 88*arctan(1/172) + 51*arctan(1/239) + 32*arctan(1/682) + 44*arctan(1/5357) + 68*arctan(
ERROR:   pi/4 = 88*arctan(1/172) + 51*arctan(1/239) + 32*arctan(1/682) + 44*arctan(1/5357) + 68*arctan(
```

Check that file exists

```
# Here
say (Dir.cwd  + %f'input.txt' -> is_file)
say (Dir.cwd  + %d'docs'      -> is_dir)

# Root
say (Dir.root + %f'input.txt' -> is_file)
say (Dir.root + %d'docs'      -> is_dir)
```

NOTE: To check only for existence, use the method *exists*

Checksumcolor

```
var ansi = frequire("Term::ANSIColor")

func colorhash(hash) {
    hash.split(2).map{|s| ansi.colored(s, "ansi" + s.hex) }.join
}

ARGF.each {|line|
    if (STDOUT.is_on_tty && (line =~ /^([[:xdigit:]]+)(.*)/)) {|m|
        say (colorhash(m[0]), m[1])
    }
    else {
        say line
    }
}
```

```
% md5sum *.sf | sf checksumcolor.sf

f8ac04c1857c  9145e1bf0fe25550d2 checksumcolor (copy).sf

f8ac04c1857c  9145e1bf0fe25550d2 checksumcolor.sf

af086941b6dc67001cd831bb1f22c3ed farey.sf

b585c25146e94df370ec4846691 1d9ae pell.sf
```

Chernick's Carmichael numbers

```
func chernick_carmichael_factors (n, m) {
    [6*m + 1, 12*m + 1, {|i| 2**i * 9*m + 1 }.map(1 .. n-2)...]
}

func is_chernick_carmichael (n, m) {
    (n == 2) ? (is_prime(6*m + 1) && is_prime(12*m + 1))
             : (is_prime(2**(n-2) * 9*m + 1) && __FUNC__(n-1, m))
}

func chernick_carmichael_number(n, callback) {
    var multiplier = (n>4 ? 2**(n-4) : 1)
    var m = (1..Inf -> first {|m| is_chernick_carmichael(n, m * multiplier) })
    var f = chernick_carmichael_factors(n, m * multiplier)
    callback(f...)
}

for n in (3..9) {
    chernick_carmichael_number(n, {|*f| say "a(#{n}) = #{f.join(' * ')}" })
}
```

Output:

```
a(3) = 7 * 13 * 19
a(4) = 7 * 13 * 19 * 37
a(5) = 2281 * 4561 * 6841 * 13681 * 27361
a(6) = 2281 * 4561 * 6841 * 13681 * 27361 * 54721
a(7) = 4681921 * 9363841 * 14045761 * 28091521 * 56183041 * 112366081 * 224732161
a(8) = 5703361 * 11406721 * 17110081 * 34220161 * 68440321 * 136880641 * 273761281 * 547522561
a(9) = 5703361 * 11406721 * 17110081 * 34220161 * 68440321 * 136880641 * 273761281 * 547522561 * 109504
```

Cheryl's birthday

```
func f(day, month) {
    Date.parse("#{day} #{month}", "%d %B")
}

var dates = [
    f(15, "May"),
    f(16, "May"),
    f(19, "May"),
    f(17, "June"),
    f(18, "June"),
    f(14, "July"),
    f(16, "July"),
    f(14, "August"),
    f(15, "August"),
    f(17, "August")
]

var filtered = dates.grep {
    dates.grep {
        dates.map{ .day }.count(.day) == 1
    }.map{ .month }.count(.month) != 1
}

var birthday = filtered.grep {
    filtered.map{ .day }.count(.day) == 1
}.group_by{ .month }.values.first_by { .len == 1 }[0]

say "Cheryl's birthday is #{birthday.fullmonth} #{birthday.day}."
```

Output:

```
Cheryl's birthday is July 16.
```

Chinese remainder theorem

```
func chinese_remainder(*n) {
    var N = n.prod
    func (*a) {
        n.range.sum { |i|
            var p = (N / n[i])
            a[i] * p.invmod(n[i]) * p
        } % N
    }
}

say chinese_remainder(3, 5, 7)(2, 3, 2)
```

Output:

```
23
```

Chinese zodiac

```
func zodiac(year) {
  var animals = %w(Rat Ox Tiger Rabbit Dragon Snake Horse Goat Monkey Rooster Dog Pig)
  var elements = %w(Wood Fire Earth Metal Water)
  var terrestrial_han = %w(子 丑 寅 卯 辰 巳 午 未 申 酉 戌 亥)
  var terrestrial_pinyin = %w(zǐ chǒu yín mǎo chén sì wǔ wèi shēn yǒu xū hài)
  var celestial_han = %w(甲 乙 丙 丁 戊 己 庚 辛 壬 癸)
  var celestial_pinyin = %w(jiǎ yǐ bǐng dīng wù jǐ gēng xīn rén gǔi)
  var aspect = %w(yang yin)

  var cycle_year = ((year-4) % 60)
  var (i2, i10, i12) = (cycle_year%2, cycle_year%10, cycle_year%12)

  (year,
   celestial_han[i10],    terrestrial_han[i12],
   celestial_pinyin[i10], terrestrial_pinyin[i12],
   elements[i10 >> 1], animals[i12], aspect[i2], cycle_year+1)
}

[1935, 1938, 1968, 1972, 1976, 2017].each { |year|
    printf("%4d: %s%s (%s-%s) %s %s; %s - year %d of the cycle\n", zodiac(year))
}
```

Output:

```
1935: 乙亥 (yǐ-hài) Wood Pig; yin - year 12 of the cycle
1938: 戊寅 (wù-yín) Earth Tiger; yang - year 15 of the cycle
1968: 戊申 (wù-shēn) Earth Monkey; yang - year 45 of the cycle
1972: 壬子 (rén-zǐ) Water Rat; yang - year 49 of the cycle
1976: 丙辰 (bǐng-chén) Fire Dragon; yang - year 53 of the cycle
2017: 丁酉 (dīng-yǒu) Fire Rooster; yin - year 34 of the cycle
```

Cholesky decomposition

```
func cholesky(matrix) {
    var chol = matrix.len.of { matrix.len.of(0) }
    for row (^matrix) {
        for col (0..row) {
            var x = matrix[row][col]
            for i (0..col) {
                x -= (chol[row][i] * chol[col][i])
            }
            chol[row][col] = (row == col ? x.sqrt : x/chol[col][col])
        }
    }
    return chol
}
```

Examples:

```
var example1 = [ [ 25, 15, -5 ],
                 [ 15, 18,  0 ],
                 [ -5,  0, 11 ] ]

say "Example 1:"
cholesky(example1).each { |row|
    say row.map {'%7.4f' % _}.join(' ')
}

var example2 = [ [ 18, 22,  54,  42],
                 [ 22, 70,  86,  62],
                 [ 54, 86, 174, 134],
                 [ 42, 62, 134, 106] ]

say "\nExample 2:"
cholesky(example2).each { |row|
    say row.map {'%7.4f' % _}.join(' ')
}
```

Output:

```
Example 1:
 5.0000  0.0000  0.0000
 3.0000  3.0000  0.0000
-1.0000  1.0000  3.0000

Example 2:
 4.2426  0.0000  0.0000  0.0000
 5.1854  6.5659  0.0000  0.0000
12.7279  3.0460  1.6497  0.0000
 9.8995  1.6246  1.8497  1.3926
```

Cipolla's algorithm

```
func cipolla(n, p) {

    legendre(n, p) == 1 || return nil

    var (a = 0, ω2 = 0)
    loop {
        ω2 = ((a*a - n) % p)
        if (legendre(ω2, p) == -1) {
            break
        }
        ++a
    }

    struct point { x, y }

    func mul(a, b) {
        point((a.x*b.x + a.y*b.y*ω2) % p, (a.x*b.y + b.x*a.y) % p)
    }

    var r = point(1, 0)
    var s = point(a, 1)

    for (var n = ((p+1) >> 1); n > 0; n >>= 1) {
        r = mul(r, s) if n.is_odd
        s = mul(s, s)
    }

    r.y == 0 ? r.x : nil
}

var tests = [
    [10, 13],
    [56, 101],
    [8218, 10007],
    [8219, 10007],
    [331575, 1000003],
    [665165880, 1000000007],
    [881398088036 1000000000039],
    [340352439146355496015833695445606502543250846432 01, 10**50 + 151],
]

for n,p in tests {
    var r = cipolla(n, p)
    if (defined(r)) {
        say "Roots of #{n} are (#{r} #{p-r}) mod #{p}"
    } else {
        say "No solution for (#{n}, #{p})"
    }
}
```

Output:

```
Roots of 10 are (6 7) mod 13
Roots of 56 are (37 64) mod 101
Roots of 8218 are (9872 135) mod 10007
No solution for (8219, 10007)
Roots of 331575 are (855842 144161) mod 1000003
Roots of 665165880 are (475131702 524868305) mod 1000000007
Roots of 881398088036 are (791399408049 208600591990) mod 1000000000039
Roots of 340352439146355496015833695445606502543250846432 01 are (825631188280903622613789939574502135 73
```

Circles of given radius through two points

```
func circles(a, b, r) {

    if (a == b) {
        if (r == 0) {
            return ['Degenerate point']
        }
        else {
            return ['Infinitely many share a point']
        }
    }

    var h = (b-a)/2

    if (r**2 < h.norm) {
        return ['Too far apart']
    }

    var l = sqrt(r**2 - h.norm)

    [1i, -1i].map {|i|
        a + h + (l*i*h / h.abs) -> round(-16)
    }
}

var input = [
    [0.1234 + 0.9876i,  0.8765 + 0.2345i, 2.0],
    [0.0000 + 2.0000i,  0.0000 + 0.0000i, 1.0],
    [0.1234 + 0.9876i,  0.1234 + 0.9876i, 2.0],
    [0.1234 + 0.9876i,  0.8765 + 0.2345i, 0.5],
    [0.1234 + 0.9876i,  0.1234 + 0.9876i, 0.0],
]

input.each {|a|
    say (a.join(', '), ': ', circles(a...).join(' and '))
}
```

Output:

```
0.1234+0.9876i, 0.8765+0.2345i, 2: 1.8631118016581891+1.9742118016581891i and -0.8632118016581891-0.752
2i, 0, 1: i and i
0.1234+0.9876i, 0.1234+0.9876i, 2: Infinitely many share a point
0.1234+0.9876i, 0.8765+0.2345i, 0.5: Too far apart
0.1234+0.9876i, 0.1234+0.9876i, 0: Degenerate point
```

Circular primes

```
func is_circular_prime(n) {
    n.is_prime || return false

    var circular = n.digits
    circular.min < circular.tail && return false

    for k in (1 ..^ circular.len) {
        with (circular.rotate(k).digits2num) {|p|
            (p.is_prime && (p >= n)) || return false
        }
    }

    return true
}

say "The first 19 circular primes are:"
say 19.by(is_circular_prime)

say "\nThe next 4 circular primes, in repunit format, are:"
{|n| (10**n - 1)/9 -> is_prob_prime }.first(4, 4..Inf).each {|n|
    say "R(#{n})"
}

say "\nRepunit testing:"
[5003, 9887, 15073, 25031, 35317, 49081].each {|n|
    var now = Time.micro
    say ("R(#{n}) -> ", is_prob_prime((10**n - 1)/9) ? 'probably prime' : 'composite',
        " (took: #{'%.3f' % Time.micro-now} sec)")
}
```

Output:

```
The first 19 circular primes are:
[2, 3, 5, 7, 11, 13, 17, 37, 79, 113, 197, 199, 337, 1193, 3779, 11939, 19937, 193939, 199933]

The next 4 circular primes, in repunit format, are:
R(19)
R(23)
R(317)
R(1031)

Repunit testing:
R(5003) -> composite (took: 0.024 sec)
R(9887) -> composite (took: 0.006 sec)
R(15073) -> composite (took: 0.389 sec)
R(25031) -> composite (took: 54.452 sec)
R(35317) -> composite (took: 0.875 sec)
```

Classes

```
class MyClass(instance_var) {
    method add(num) {
        instance_var += num
    }
}

var obj = MyClass(3)    # `instance_var` will be set to 3
obj.add(5)              # calls the add() method
say obj.instance_var    # prints the value of `instance_var`: 8
```

Closest-pair problem

```
func dist_squared(a, b) {
    sqr(a[0] - b[0]) + sqr(a[1] - b[1])
}

func closest_pair_simple(arr) {
    arr.len < 2 && return Inf
    var (a, b, d) = (arr[0, 1], dist_squared(arr[0,1]))
    arr.clone!
    while (arr) {
        var p = arr.pop
        for l in arr {
            var t = dist_squared(p, l)
            if (t < d) {
                (a, b, d) = (p, l, t)
            }
        }
    }
    return(a, b, d.sqrt)
}

func closest_pair_real(rx, ry) {
    rx.len <= 3 && return closest_pair_simple(rx)

    var N = rx.len
    var midx = (ceil(N/2)-1)
    var (PL, PR) = rx.part(midx)

    var xm = rx[midx][0]

    var yR = []
    var yL = []

    for item in ry {
        (item[0] <= xm ? yR : yL) << item
    }

    var (al, bl, dL) = closest_pair_real(PL, yR)
    var (ar, br, dR) = closest_pair_real(PR, yL)

    al == Inf && return (ar, br, dR)
    ar == Inf && return (al, bl, dL)

    var (m1, m2, dmin) = (dR < dL ? [ar, br, dR]...
                                  : [al, bl, dL]...)

    var yS = ry.grep { |a| abs(xm - a[0]) < dmin }

    var (w1, w2, closest) = (m1, m2, dmin)
    for i in (0 ..^ yS.end) {
```

```
        for k in (i+1 .. yS.end) {
            yS[k][1] - yS[i][1] < dmin || break
            var d = dist_squared(yS[k], yS[i]).sqrt
            if (d < closest) {
                (w1, w2, closest) = (yS[k], yS[i], d)
            }
        }
    }

    return (w1, w2, closest)
}

func closest_pair(r) {
    var ax = r.sort_by { |a| a[0] }
    var ay = r.sort_by { |a| a[1] }
    return closest_pair_real(ax, ay);
}

var N = 100
var points = N.of { [1.rand*20 - 10, 1.rand*20 - 10] }
var (af, bf, df) = closest_pair(points)
say "#{df} at (#{af.join(' ')}), (#{bf.join(' ')})"
```

Closures/Value capture

```
var f = (
    10.of {|i| func(j){i * j} }
)

9.times { |j|
    say f[j](j)
}
```

Output:

```
0
1
4
9
16
25
36
49
64
```

Starting from i=1:

```
var f = (1..10).map { |i|
    func(j){i * j}
}

for j (1..9) {
    say f[j-1](j)
}
```

Output:

```
1
4
9
16
25
36
49
64
81
```

Collect and sort square numbers in ascending order from three lists

```
var lists = [
  [3,4,34,25,9,12,36,56,36],
  [2,8,81,169,34,55,76,49,7],
  [75,121,75,144,35,16,46,35],
]

say lists.flat.grep{.is_square}.sort
```

Output:

```
[4, 9, 16, 25, 36, 36, 49, 81, 121, 144, 169]
```

Collections

Arrays are ordered, integer-indexed collections of any object.

```
# creating an empty array and adding values
var a = []          #=> []
a[0] = 1            #=> [1]
a[3] = "abc"        #=> [1, nil, nil, "abc"]
a << 3.14           #=> [1, nil, nil, "abc", 3.14]
```

A Hash is a dictionary-like collection of unique keys and their values. Also called associative arrays, they are similar to Arrays, but where an Array uses integers as its index, a Hash allows you to use any object type, which is automatically converted into a String.

```
# creating an empty hash
var h = Hash()    #=> Hash()
h{:foo} = 1       #=> Hash("foo"=>1)
h{:bar} = 2.4     #=> Hash("foo"=>1, "bar"=>2.4)
h{:bar} += 3      #=> Hash("foo"=>1, "bar"=>5.4)
```

A Pair is an array-like collection, but restricted only to two elements.

```
# create a simple pair
var p = Pair('a', 'b')
say p.first;            #=> 'a'
say p.second;           #=> 'b'

# create a pair of pairs
var pair = 'foo':'bar':'baz':();   # => Pair('foo', Pair('bar', Pair('baz', nil)))

# iterate over the values of a pair of pairs
loop {
    say pair.first;               #=> 'foo', 'bar', 'baz'
    pair = pair.second;
    pair == nil && break;
}
```

A Struct is a convenient way to bundle a number of attributes together.

```
# creating a struct
struct Person {
    String name,
    Number age,
    String sex
}

var a = Person("John Smith", 41, :man)

a.age += 1                  # increment age
a.name = "Dr. #{a.name}"    # update name

say a.name          #=> "Dr. John Smith"
say a.age           #=> 42
say a.sex           #=> "man"
```

Color quantization

```
require('Image::Magick')

func quantize_image(n = 16, input, output='output.png') {
    var im = %O<Image::Magick>.new
    im.Read(input)
    im.Quantize(colors => n, dither => 1)    # 1 = None
    im.Write(output)
}

quantize_image(input: 'Quantum_frog.png')
```

Color wheel

```
require('Imager')

var (width, height) = (300, 300)
var center = Complex(width/2 , height/2)

var img = %O<Imager>.new(xsize => width, ysize => height)

for y=(^height), x=(^width) {
    var vector    = (center - x - y.i)
    var magnitude = (vector.abs * 2 / width)
    var direction = ((Num.pi + atan2(vector.real, vector.imag)) / Num.tau)
    img.setpixel(x => x, y => y,
        color => Hash(hsv => [360*direction, magnitude, magnitude < 1 ? 1 : 0])
    )
}

img.write(file => 'color_wheel.png')
```

Output image

Colour bars/Display

```
require('GD')

var colors = Hash(
              white   => [255, 255, 255],
              red     => [255, 0,   0],
              green   => [0,   255, 0],
              blue    => [0,   0,   255],
              magenta => [255, 0,   255],
              yellow  => [255, 255, 0],
              cyan    => [0,   255, 255],
              black   => [0,   0,   0],
             )

var barwidth = 160/8
var image    = %O<GD::Image>.new(160, 100)
var start    = 0

colors.values.each { |rgb|
    var paintcolor = image.colorAllocate(rgb...)
    image.filledRectangle(start * barwidth, 0, start*barwidth + barwidth - 1, 99, paintcolor)
    start++
}

%f'colorbars.png'.open('>:raw').print(image.png)
```

Colour pinstripe/Display

```
require('GD')

func pinstripes(width = 1280, height = 720) {

    var im = %O<GD::Image>.new(width, height)
    var colors = [0, 255].variations_with_repetition(3)

    var paintcolors = colors.shuffle.map {|rgb|
        im.colorAllocate(rgb...)
    }

    var starty     = 0
    var barheight  = height//4

    for barwidth in (1..4) {
        for (
            var(startx = 0, colorindex = 0);
            startx + barwidth <= width;
            startx += barwidth
        ) {
            im.filledRectangle(startx, starty, startx+barwidth,
                starty + barheight - 1, paintcolors[colorindex++ % 8])
        }
        starty += barheight
    }

    return im
}

File('pinstripes.png').write(pinstripes().png, :raw)
```

Combinations

Built-in

```
combinations(5, 3, {|*c| say c })
```

Recursive

```
func combine(n, set) {

    set.len || return []
    n == 1  && return set.map{[_]}

    var (head, result)
    head   = set.shift
    result = combine(n-1, [set...])

    for subarray in result {
        subarray.prepend(head)
    }

    result + combine(n, set)
}

combine(3, @^5).each {|c| say c }
```

Iterative

```
func forcomb(callback, n, k) {

    if (k == 0) {
        callback([])
        return()
    }

    if (k<0 || k>n || n==0) {
        return()
    }

    var c = @^k

    loop {
        callback([c...])
        c[k-1]++ < n-1 && next
        var i = k-2
        while (i>=0 && c[i]>=(n-(k-i))) {
            --i
        }
        i < 0 && break
        c[i]++
        while (++i < k) {
            c[i] = c[i-1]+1
        }
    }

    return()
}

forcomb({|c| say c }, 5, 3)
```

Output

```
[0, 1, 2]
[0, 1, 3]
[0, 1, 4]
[0, 2, 3]
[0, 2, 4]
[0, 3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]
```

Combinations and permutations

```
func P(n, k) { n! / ((n-k)!) }
func C(n, k) { binomial(n, k) }

class Logarithm(value) {
    method to_s {
        var e = int(value/10.log)
        "%.8fE%+d" % (exp(value - e*10.log), e)
    }
}

func lstirling(n) {
    n < 10 ? (lstirling(n+1) - log(n+1))
           : (0.5*log(2*Num.pi*n) + n*log(n/Num.e + 1/(12*Num.e*n)))
}

func P_approx(n, k) {
    Logarithm((lstirling(n) - lstirling(n -k)))
}

func C_approx(n, k) {
    Logarithm((lstirling(n) - lstirling(n -k) - lstirling(k)))
}

say "=> Exact results:"
for n (1..12) {
    var p = n//3
    say "P(#{n}, #{p}) = #{P(n, p)}"
}

for n (10..60 `by` 10) {
    var p = n//3
    say "C(#{n}, #{p}) = #{C(n, p)}"
}

say '';
say "=> Floating point approximations:"
for n ([5, 50, 500, 1000, 5000, 15000]) {
    var p = n//3
    say "P(#{n}, #{p}) = #{P_approx(n, p)}"
}

for n (100..1000 `by` 100) {
    var p = n//3
    say "C(#{n}, #{p}) = #{C_approx(n, p)}"
}
```

Output:

```
=> Exact results:
P(1, 0) = 1
P(2, 0) = 1
P(3, 1) = 3
P(4, 1) = 4
P(5, 1) = 5
P(6, 2) = 30
P(7, 2) = 42
P(8, 2) = 56
P(9, 3) = 504
P(10, 3) = 720
P(11, 3) = 990
P(12, 4) = 11880
C(10, 3) = 120
C(20, 6) = 38760
C(30, 10) = 30045015
C(40, 13) = 12033222880
C(50, 16) = 4923689695575
C(60, 20) = 4191844505805495

=> Floating point approximations:
P(5, 1) = 5.00000000E+0
P(50, 16) = 1.03017326E+26
P(500, 166) = 3.53487492E+434
P(1000, 333) = 5.96932629E+971
P(5000, 1666) = 6.85674576E+6025
P(15000, 5000) = 9.64985399E+20469
C(100, 33) = 2.94692433E+26
C(200, 66) = 7.26975256E+53
C(300, 100) = 4.15825147E+81
C(400, 133) = 1.25794868E+109
C(500, 166) = 3.92602839E+136
C(600, 200) = 2.50601778E+164
C(700, 233) = 8.10320356E+191
C(800, 266) = 2.64562336E+219
C(900, 300) = 1.74335637E+247
C(1000, 333) = 5.77613455E+274
```

Combinations with repetitions

```
func cwr (n, l, a = []) {
    n>0 ? (^l -> map {|k| __FUNC__(n-1, l.slice(k), [a..., l[k]]) }) : a
}

cwr(2, %w(iced jam plain)).each {|a|
    say a.map{ .join(' ') }.join("\n")
}
```

Also built-in:

```
%w(iced jam plain).combinations_with_repetition(2, {|*a|
    say a.join(' ')
})
```

Output:

```
iced iced
iced jam
iced plain
jam jam
jam plain
plain plain
```

Efficient counting of the total number of combinations with repetition:

```
func cwr_count (n, m) { binomial(n + m - 1, m) }
printf("\nThere are %s ways to pick 7 out of 10 with repetition\n", cwr_count(10, 7))
```

Output:

```
There are 11440 ways to pick 7 out of 10 with repetition
```

Comma quibbling

```
func comma_quibbling(words) {
    '{' + ([words.ft(0, -2).join(', ')]-['']  + [words.last] -> join(' and ')) + '}';
}

[<>, <ABC>, <ABC DEF>, <ABC DEF G H>].each { |w|
    say comma_quibbling(w);
}
```

Output:

```
{}
{ABC}
{ABC and DEF}
{ABC, DEF, G and H}
```

Command-line arguments

Command line arguments are available in the ARGV array.

```
say ARGV
```

Output:

```
% myprog -c "alpha beta" -h "gamma"
['-c', 'alpha beta', '-h', 'gamma']
```

Comments

Single line comment

```
# this is commented
```

These may also be at the end of a line

```
var i = 1; # this is the comment part
```

Embedded comments

```
var distance #`{in meters} = (30 #`{meters} * 100 #`{seconds});
say distance; # prints: 3000
```

Multi-line comments

```
/*
    This is a multi-line comment
*/
```

Compare a list of strings

Short-circuiting:

```
1..arr.end -> all{ arr[0] == arr[_] }   # all equal
1..arr.end -> all{ arr[_-1] < arr[_] }  # strictly ascending
```

Non short-circuiting:

```
arr.uniq.len == 1      # all equal
arr == arr.uniq.sort   # strictly ascending
```

Compile-time calculation

The compile-time evaluation is limited at a constant expression, which cannot refer at any other user-defined data, such as variables or functions.

```
define n = (10!)
say n
```

or:

```
define n = (func(n){ n > 0 ? __FUNC__(n-1)*n : 1 }(10))
say n
```

Composite numbers k with no single digit factors whose factors are all substrings of k

```
var e = Enumerator({|f|

    var c = (9.primorial)
    var a = (1..c -> grep { .is_coprime(c) })

    loop {
        var n = a.shift

        a.push(n + c)
        n.is_composite || next

        f(n) if n.factor.all {|p| Str(n).contains(p) }
    }
})

var count = 10

e.each {|n|
    say n
    break if (--count <= 0)
}
```

Output:

```
15317
59177
83731
119911
183347
192413
1819231
2111317
2237411
3129361
```

Compound data type

```
struct Point {x, y}
var point = Point(1, 2)
say point.y                #=> 2
```

Concatenate two primes is also prime

```
var upto = 100
var arr  = upto.primes
var base = 10

arr = arr.map {|p|
    var d = p.digits(base);
    arr.map {|q| [q.digits(base)..., d...].digits2num(base) }.grep { .is_prime }
}.flat.uniq.sort

say "Concatenated primes from primes p,q <= #{upto}:"

arr.each_slice(10, {|*a|
    say a.map { '%6s' % _ }.join(' ')
})

say "\nFound #{arr.len} such concatenated primes."
```

Output:

```
Concatenated primes from primes p,q <= 100:
    23     37     53     73    113    137    173    193    197    211
   223    229    233    241    271    283    293    311    313    317
   331    337    347    353    359    367    373    379    383    389
   397    433    523    541    547    571    593    613    617    673
   677    719    733    743    761    773    797    977   1117   1123
  1129   1153   1171   1319   1361   1367   1373   1723   1741   1747
  1753   1759   1783   1789   1913   1931   1973   1979   1997   2311
  2341   2347   2371   2383   2389   2917   2953   2971   3119   3137
  3167   3719   3761   3767   3779   3797   4111   4129   4153   4159
  4337   4373   4397   4723   4729   4759   4783   4789   5323   5347
  5923   5953   6113   6131   6143   6173   6197   6719   6737   6761
  6779   7129   7159   7331   7919   7937   8311   8317   8329   8353
  8389   8923   8929   8941   8971   9719   9743   9767

Found 128 such concatenated primes.
```

Concurrent computing

A very basic threading support is provided by the *Block.fork()* method:

```
<Enjoy Rosetta Code>            \
    .map{|str| {say str}.fork } \
    .map{|thr| thr.wait }
```

Output:

```
Enjoy
Code
Rosetta
```

Conjugate transpose

```
func is_Hermitian (Array m, Array t) -> Bool { m == t }
```

```
func mat_mult (Array a, Array b, Number ε = -3) {
    var p = []
    for r, c in (^a ~X ^b[0]) {
        for k in (^b) {
            p[r][c] := 0 += (a[r][k] * b[k][c]) -> round!(ε)
        }
    }
    return p
}

func mat_trans (Array m) {
    var r = []
    for i,j in (^m ~X ^m[0]) {
        r[j][i] = m[i][j]
    }
    return r
}

func mat_ident (Number n) {
    ^n -> map {|i|
        [i.of(0)..., 1, (n - i - 1).of(0)...]
    }
}

func is_Normal (Array m, Array t) -> Bool {
    mat_mult(m, t) == mat_mult(t, m)
}

func is_Unitary (Array m, Array t) -> Bool {
    mat_mult(m, t) == mat_ident(m.len)
}

func say_it (Array a) {
    a.each {|b|
        b.map { "%9s" % _ }.join(' ').say
    }
}

[
    [
       [   1, 1+1i, 2i],
       [1-1i,    5, -3],
       [0-2i,   -3,  0]
    ],
    [
       [1, 1, 0],
       [0, 1, 1],
       [1, 0, 1]
    ],
    [
       [0.707 ,   0.707,  0],
       [0.707i, -0.707i,  0],
       [0     ,       0,  1i]
    ]
].each { |m|
    say "\nMatrix:"
    say_it(m)
    var t = mat_trans(m.map{.map{.conj}})
    say "\nTranspose:"
    say_it(t)
    say "Is Hermitian?\t#{is_Hermitian(m, t)}"
    say "Is Normal?\t#{is_Normal(m, t)}"
    say "Is Unitary?\t#{is_Unitary(m, t)}"
}
```

Output:

```
Matrix:
        1       1+i        2i
      1-i         5        -3
      -2i        -3         0

Transpose:
        1       1+i        2i
      1-i         5        -3
      -2i        -3         0
Is Hermitian?   true
Is Normal?      true
Is Unitary?     false

Matrix:
        1         1         0
        0         1         1
        1         0         1

Transpose:
        1         0         1
        1         1         0
        0         1         1
Is Hermitian?   false
Is Normal?      true
Is Unitary?     false

Matrix:
    0.707     0.707         0
   0.707i   -0.707i         0
        0         0         i

Transpose:
    0.707   -0.707i         0
    0.707    0.707i         0
        0         0        -i
Is Hermitian?   false
Is Normal?      true
Is Unitary?     true
```

Consecutive primes with ascending or descending differences

```
func runs(f, arr) {

    var run = 0
    var diff = 0
    var diffs = []

    arr.each_cons(2, {|p1,p2|
        var curr_diff = (p2 - p1)
        f(curr_diff, diff) ? ++run : (run = 1)
        diff = curr_diff
        diffs << run
    })

    var max  = diffs.max
    var runs = []

    diffs.indices_by { _ == max }.each {|i|
        runs << arr.slice(i - max + 1, i + 1)
    }

    return runs
}

var limit = 1e6
var primes = limit.primes

say "Longest run(s) of ascending prime gaps up to #{limit.commify}:"
say runs({|a,b| a > b }, primes).join("\n")

say "\nLongest run(s) of descending prime gaps up to #{limit.commify}:"
say runs({|a,b| a < b }, primes).join("\n")
```

Output:

```
Longest run(s) of ascending prime gaps up to 1,000,000:
[128981, 128983, 128987, 128993, 129001, 129011, 129023, 129037]
[402581, 402583, 402587, 402593, 402601, 402613, 402631, 402691]
[665111, 665113, 665117, 665123, 665131, 665141, 665153, 665177]

Longest run(s) of descending prime gaps up to 1,000,000:
[322171, 322193, 322213, 322229, 322237, 322243, 322247, 322249]
[752207, 752251, 752263, 752273, 752281, 752287, 752291, 752293]
```

Constrained genericity

```
class FoodBox(*food { .all { .respond_to(:eat) } }) { }

class Fruit { method eat { ... } }
class Apple < Fruit {  }

say FoodBox(Fruit(), Apple()).dump  #=> FoodBox(food: [Fruit(), Apple()])
say FoodBox(Apple(), "foo")         #!> ERROR: class `FoodBox` !~ (Apple, String)
```

Constrained random points on a circle

Generates an EPS file.

```
var points = []
while (points.len < 100) {
    var (x, y) = 2.of{ 30.irand - 15 }...
    var r2 = (x**2 + y**2)
    if ((r2 >= 100) && (r2 <= 225)) {
        points.append([x, y])
    }
}

print <<'HEAD'
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox 0 0 400 400
200 200 translate 10 10 scale
0 setlinewidth
1 0 0 setrgbcolor
0 0 10 0 360 arc stroke
0 0 15 360 0 arcn stroke
0 setgray
/pt { .1 0 360 arc fill } def
HEAD

points.each { |pt| say "#{pt.join(' ')} pt" }
print '%%EOF'
```

Continued fraction

```
func continued_fraction(a, b, f, n = 1000, r = 1) {
    f(func (r) {
        r < n ? (a(r) / (b(r) + __FUNC__(r+1))) : 0
    }(r))
}

var params = Hash(
    "φ"  => [ { 1 }, { 1 }, { 1 + _ } ],
    "√2" => [ { 1 }, { 2 }, { 1 + _ } ],
    "e"  => [ { _ }, { _ }, { 1 + 1/_ } ],
    "π"  => [ { (2*_ - 1)**2 }, { 6 }, { 3 + _ } ],
    "τ"  => [ { _**2 }, { 2*_ + 1 }, { 8 / (1 + _) } ],
)

for k in (params.keys.sort) {
    printf("%2s ≈ %s\n", k, continued_fraction(params{k}...))
}
```

Output:

```
 e ≈ 2.71828182845904523536028747135266249775724709369
 π ≈ 3.14159265383979292596359650286939597045138933078
 τ ≈ 6.28318530717958647692528676655900576839433879875
 φ ≈ 1.61803398874989484820458683436563811772030917981
√2 ≈ 1.41421356237309504880168872420969807856967187538
```

Continued fraction/Arithmetic/Construct from rational number

```
func r2cf(num, den) {
    func() {
        den || return nil
        var q = num//den
        (num, den) = (den, num - q*den)
        return q
    }
}

func showcf(f) {
    print "["
    var n = f()
    print "#{n}" if defined(n)
    print "; #{n}" while defined(n = f())
    print "]\n"
}

[
    [1/2, 3/1, 23/8, 13/11, 22/7, -151/77],
    [14142/10000, 141421/100000, 1414214/1000000, 14142136/10000000],
    [314285714/100000000],
].each { |seq|
    seq.each { |r| showcf(r2cf(r.nude)) }
    print "\n"
}
```

Output:

```
[0; 2]
[3]
[2; 1; 7]
[1; 5; 2]
[3; 7]
[-1; -1; -24; -1; -2]

[1; 2; 2; 2; 2; 2; 1; 1; 29]
[1; 2; 2; 2; 2; 2; 2; 3; 1; 1; 3; 1; 7; 2]
[1; 2; 2; 2; 2; 2; 2; 2; 3; 6; 1; 2; 1; 12]
[1; 2; 2; 2; 2; 2; 2; 2; 2; 2; 6; 1; 2; 4; 1; 1; 2]

[3; 7; 7142857]
```

Convert decimal number to rational

By default, literal numbers are represented in rational form:

```
say 0.75.as_frac          #=> 3/4
say 0.518518.as_frac      #=> 259259/500000
say 0.9054054.as_frac     #=> 4527027/5000000
```

Additionally, *Num(str)* can be used for parsing a decimal expansion into rational form:

```
'0.9054054 0.518518 0.75'.split.each { |str|
    say Num(str).as_frac
}
```

Output:

```
4527027/5000000
259259/500000
3/4
```

For rational approximations, the Number *.rat_approx* method can be used:

```
say 0.518518.rat_approx.as_frac    #=> 14/27
say 0.9054054.rat_approx.as_frac   #=> 67/74
```

Convert seconds to compound duration

```
func polymod(n, *divs) {
    gather {
        divs.each { |i|
            var m = take(n % i)
            (n -= m) /= i
        }
        take(n)
    }
}

func compound_duration(seconds) {
    (polymod(seconds, 60, 60, 24, 7) ~Z <sec min hr d wk>).grep { |a|
        a[0] > 0
    }.reverse.map{.join(' ')}.join(', ')
}

[7259, 86400, 6000000].each { |s|
    say "#{'%7d' % s} sec  =  #{compound_duration(s)}"
}
```

Output:

```
   7259 sec  =  2 hr, 59 sec
  86400 sec  =  1 d
6000000 sec  =  9 wk, 6 d, 10 hr, 40 min
```

Convex hull

```
class Point(Number x, Number y) {
    method to_s {
        "(#{x}, #{y})"
    }
}

func ccw (Point a, Point b, Point c) {
    (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x)
}

func tangent (Point a, Point b) {
    (b.x - a.x) / (b.y - a.y)
}
```

```
func graham_scan (*coords) {

    ## sort points by y, secondary sort on x
    var sp = coords.map { |a|
        Point(a...)
    }.sort { |a,b|
        (a.y <=> b.y) ||
        (a.x <=> b.x)
    }

    # need at least 3 points to make a hull
    if (sp.len < 3) {
        return sp
    }

    # first point on hull is minimum y point
    var h = [sp.shift]

    # re-sort the points by angle, secondary on x
    sp = sp.map_kv { |k,v|
        Pair(k, [tangent(h[0], v), v.x])
    }.sort { |a,b|
        (b.value[0] <=> a.value[0]) ||
        (a.value[1] <=> b.value[1])
    }.map { |a|
        sp[a.key]
    }

    # first point of re-sorted list is guaranteed to be on hull
    h << sp.shift

    # check through the remaining list making sure that
    # there is always a positive angle
    sp.each { |point|
        loop {
            if (ccw(h.last(2)..., point) >= 0) {
                h << point
                break
            } else {
                h.pop
            }
        }
    }

    return h
}

var hull = graham_scan(
    [16, 3], [12,17], [ 0, 6], [-4,-6], [16, 6], [16,-7], [16,-3],
    [17,-4], [ 5,19], [19,-8], [ 3,16], [12,13], [ 3,-4], [17, 5],
    [-3,15], [-3,-9], [ 0,11], [-9,-3], [-4,-2], [12,10])

say("Convex Hull (#{hull.len} points): ", hull.join(" "))

hull = graham_scan(
    [16, 3], [12,17], [ 0, 6], [-4,-6], [16, 6], [16,-7], [16,-3],
    [17,-4], [ 5,19], [19,-8], [ 3,16], [12,13], [ 3,-4], [17, 5],
    [-3,15], [-3,-9], [ 0,11], [-9,-3], [-4,-2], [12,10], [14,-9], [1,-9])

say("Convex Hull (#{hull.len} points): ", hull.join(" "))
```

Output:

```
Convex Hull (7 points): (-3, -9) (19, -8) (17, 5) (12, 17) (5, 19) (-3, 15) (-9, -3)
Convex Hull (9 points): (-3, -9) (1, -9) (14, -9) (19, -8) (17, 5) (12, 17) (5, 19) (-3, 15) (-9, -3)
```

Conway's Game of Life

```
var w = Num(`tput cols`)
var h = Num(`tput lines`)
var r = "\033[H"

var dirs = [[-1,-1], [-1, 0], [-1, 1], [ 0,-1],
            [ 0, 1], [ 1,-1], [ 1, 0], [ 1, 1]]

var universe = h.of { w.of {1.rand < 0.1} }

func iterate {
    var new = h.of { w.of(false) }
    static rx = (^h ~X ^w)
    for i,j in rx {
        var neighbor = 0
        for y,x in (dirs.map {|dir| dir »+« [i, j] }) {
            universe[y % h][x % w] && ++neighbor
            neighbor > 3 && break
        }
        new[i][j] = (universe[i][j]
                        ? (neighbor==2 || neighbor==3)
                        : (neighbor==3))
    }
    universe = new
}

STDOUT.autoflush(true)

loop {
    print r
    say universe.map{|row| row.map{|cell| cell ? '#' : ' '}.join }.join("\n")
    iterate()
}
```

Coprime triplets

```
func coprime_triplets(callback) {

    var (
        list = [1,2],
        a = 1,
        b = 2,
        k = 3,
        seen = Set()
    )

    loop {
        for (var n = k; true; ++n) {
            if (!seen.has(n) && is_coprime(n, a) && is_coprime(n, b)) {

                list << n
                seen << n

                callback(list) && return list

                (a, b) = (b, n)

                while (seen.has(k)) {
                    seen.remove(k++)
                }

                break
            }
        }
    }
}

say "Coprime triplets before first term is > 50:"
coprime_triplets({|list|
    list.tail >= 50
}).first(-1).slices(10).each { .«%« '%4d' -> join(' ').say }

say "\nLeast Coprime triplets that encompass 1 through 50:"
coprime_triplets({|list|
    list.sort.first(50) == @(1..50)
}).slices(10).each { .«%« '%4d' -> join(' ').say }

say "\n1001st through 1050th Coprime triplet:"
coprime_triplets({|list|
    list.len == 1050
}).last(50).slices(10).each { .«%« '%4d' -> join(' ').say }
```

Output:

```
Coprime triplets before first term is > 50:
   1    2    3    5    4    7    9    8   11   13
   6   17   19   10   21   23   16   15   29   14
  25   27   22   31   35   12   37   41   18   43
  47   20   33   49   26   45

Least Coprime triplets that encompass 1 through 50:
   1    2    3    5    4    7    9    8   11   13
   6   17   19   10   21   23   16   15   29   14
  25   27   22   31   35   12   37   41   18   43
  47   20   33   49   26   45   53   28   39   55
  32   51   59   38   61   63   34   65   57   44
  67   69   40   71   73   24   77   79   30   83
  89   36   85   91   46   75   97   52   81   95
  56   87  101   50   93  103   58   99  107   62
 105  109   64  111  113   68  115  117   74  119
 121   48  125  127   42

1001st through 1050th Coprime triplet:
 682 1293 1361  680 1287 1363  686 1299 1367  688
1305 1369  692 1311 1373  694 1317 1375  698 1323
1381  704 1329 1379  706 1335 1387  716 1341 1385
 712 1347 1391  700 1353 1399  710 1359 1393  718
1371 1397  722 1365 1403  724 1377 1405  728 1383
```

Coprimes

```
var pairs = [[21,15],[17,23],[36,12],[18,29],[60,15]]
say "The following pairs of numbers are coprime:"
pairs.grep { .gcd == 1 }.each { .say }
```

Output:

```
The following pairs of numbers are coprime:
[17, 23]
[18, 29]
```

Copy a string

```
var original = "hello"               # new String object
var reference = original             # points at the original object
var copy1 = String(original)         # creates a new String object
var copy2 = original+''              # ==//==
```

Count in factors

```
class Counter {
    method factors(n, p=2) {
        var a = gather {
            while (n >= p*p) {
                while (p `divides` n) {
                    take(p)
                    n //= p
                }
                p = self.next_prime(p)
            }
        }
        (n > 1 || a.is_empty) ? (a << n) : a
    }

    method is_prime(n) {
        self.factors(n).len == 1
    }

    method next_prime(p) {
        do {
            p == 2 ? (p = 3) : (p+=2)
        } while (!self.is_prime(p))
        return p
    }
}

for i in (1..100) {
    say "#{i} = #{Counter().factors(i).join(' × ')}"
}
```

Count in octal

```
var i = 0;
loop { say i++.as_oct }
```

Count occurrences of a substring

Built-in:

```
say "the three truths".count("th")
say "abababababab".count("abab")
```

User-created function:

```
func countSubstring(s, ss) {
    var re = Regex(ss.escape, 'g')      # 'g' for global
    var counter = 0
    while (s =~ re) { ++counter }
    return counter
}

say countSubstring("the three truths","th")
say countSubstring("abababababab","abab")
```

Output:

```
3
2
```

Count the coins

```
func cc(_)                { 0 }
func cc({ .is_neg  }, *_) { 0 }
func cc({ .is_zero }, *_) { 1 }

func cc(amount, first, *rest) is cached {
    cc(amount, rest...) + cc(amount - first, first, rest...);
}

func cc_optimized(amount, *rest) {
    cc(amount, rest.sort_by{|v| -v }...);
}

var x = cc_optimized(100, 1, 5, 10, 25);
say "Ways to change $1 with common coins: #{x}";

var y = cc_optimized(1000 * 100, 1, 5, 10, 25, 50, 100);
say "Ways to change $1000 with addition of less common coins: #{y}";
```

Output:

```
Ways to change $1 with common coins: 242
Ways to change $1000 with addition of less common coins: 13398445413854501
```

Cousin primes

```
var limit = 1000
var pairs = (limit-5).primes.map { [_, _+4] }.grep { .tail.is_prime }

say "Cousin prime pairs whose elements are less than #{limit.commify}:"
say pairs
say "\n#{pairs.len} pairs found"
```

Output:

```
Cousin prime pairs whose elements are less than 1,000:
[[3, 7], [7, 11], [13, 17], [19, 23], [37, 41], [43, 47], [67, 71], [79, 83], [97, 101], [103, 107], [1

41 pairs found
```

Cramer's rule

```
func cramers_rule(A, terms) {
    gather {
        for i in ^A {
            var Ai = A.map{.map{_}}
            for j in ^terms {
                Ai[j][i] = terms[j]
            }
            take(Ai.det)
        }
    } »/» A.det
}

var matrix = [
    [2, -1,  5,  1],
    [3,  2,  2, -6],
    [1,  3,  3, -1],
    [5, -2, -3,  3],
]

var free_terms = [-3, -32, -47, 49]
var (w, x, y, z) = cramers_rule(matrix, free_terms)...

say "w = #{w}"
say "x = #{x}"
say "y = #{y}"
say "z = #{z}"
```

Output:

```
w = 2
x = -12
y = -4
z = 1
```

Create a file

```
# Here
%f'output.txt' -> create
%d'docs'       -> create

# Root dir
Dir.root + %f'output.txt' -> create
Dir.root + %d'docs'       -> create
```

Create a two-dimensional array at runtime

```
func make_matrix(x, y) {
    y.of { x.of(0) }
}

var y = read("rows: ", Number)
var x = read("cols: ", Number)

var matrix = make_matrix(x, y)    # create the matrix
matrix[y/2][x/2] = 1              # write something inside it
say matrix                         # display the matrix
```

Output:

```
rows: 3
cols: 4
[[0, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 0]]
```

Create an HTML table

```
class HTML {
    method _attr(Hash h) {
        h.keys.sort.map {|k| %Q' #{k}="#{h{k}}"' }.join
    }

    method _tag(Hash h, name, value) {
        "<#{name}" + self._attr(h) + '>' + value + "</#{name}>"
    }

    method table(Hash h, *data) { self._tag(h, 'table', data.join) }
    method table(*data)         { self.table(Hash(), data...) }
}

class Table < HTML {
    method th(Hash h, value) { self._tag(h, 'th', value) }
    method th(value)         { self.th(Hash(), value) }

    method tr(Hash h, *rows) { self._tag(h, 'tr', rows.join) }
    method tr(*rows)         { self.tr(Hash(), rows...) }

    method td(Hash h, value) { self._tag(h, 'td', value) }
    method td(value)         { self.td(Hash(), value) }
}

var header = %w(  X Y Z)
var rows = 5

var html = HTML()
var table = Table()

say html.table(
    # attributes
    Hash(
        cellspacing => 4,
        style => "text-align:right; border: 1px solid;"
     ),

    # header
    table.tr(header.map{|elem| table.th(elem)}...),

    # rows
    (1..rows).map { |i|
        table.tr(
            table.td(:(align => 'right'), i),
            (header.len - 1).of {
                table.td(Hash(align => 'right'), 10000.irand)
            }...
        )
    }...
)
```

Output (tidied afterwards):

```
<table cellspacing="4" style="text-align:right; border: 1px solid;">
<tr><th> </th><th>X</th><th>Y</th><th>Z</th></tr>
<tr>
  <td align="right">1</td>
  <td align="right">2308</td>
  <td align="right">6448</td>
  <td align="right">2614</td>
</tr>
<tr>
  <td align="right">2</td>
  <td align="right">8830</td>
  <td align="right">553</td>
  <td align="right">5647</td>
</tr>
<tr>
  <td align="right">3</td>
  <td align="right">9636</td>
  <td align="right">5922</td>
  <td align="right">6384</td>
</tr>
<tr>
  <td align="right">4</td>
  <td align="right">9122</td>
  <td align="right">4832</td>
  <td align="right">8813</td>
</tr>
<tr>
  <td align="right">5</td>
  <td align="right">3331</td>
  <td align="right">5528</td>
  <td align="right">701</td>
</tr>
</table>
```

CSV data manipulation

For simple files we can use the *split* method.

```
# Read
var csvfile = %f'data.csv';
var fh = csvfile.open_r;
var header = fh.line.trim_end.split(',');
var csv = fh.lines.map { .trim_end.split(',').map{.to_num} }
fh.close;

# Write
var out = csvfile.open_w;
out.say([header..., 'SUM'].join(','));
csv.each { |row| out.say([row..., row.sum].join(',')) };
out.close;
```

For complex files, the *Text::CSV* library is recommended.

```
var csv = require('Text::CSV').new(
    Hash(eol => "\n")
);

# Open
var csvfile = %f'data.csv';
var fh = csvfile.open_r;

# Read
var rows = [];
var header = csv.getline(fh);
while (var row = csv.getline(fh)) {
    rows.append(row.map{.to_num});
}

# Process
header.append('SUM');
rows.each { |row| row.append(row.sum) }

# Write
var out = csvfile.open_w;
[header, rows...].each { |row|
    csv.print(out, row);
}
```

CSV to HTML translation

```
func escape(str) { str.trans(« & < > », « &amp; &lt; &gt; ») }
func tag(t, d)   { "<#{t}>#{d}</#{t}>" }

func csv2html(str) {

    var template = <<-'EOT'
    <!DOCTYPE html>
    <html>
    <head><title>Some Text</title></head>
    <body><table>
    %s
    </table></body></html>
    EOT

    template.sprintf(escape(str).lines.map{ |line|
            tag('tr', line.split(',').map{|cell| tag('td', cell) }.join)
        }.join("\n")
    )
}

var str = <<'EOT';
Character,Speech
The multitude,The messiah! Show us the messiah!
Brians mother,<angry>Now you listen here! He's not the messiah; he's a very naughty boy! Now go
away!</angry>
The multitude,Who are you?
Brians mother,I'm his mother; that's who!
The multitude,Behold his mother! Behold his mother!
EOT

print csv2html(str)
```

```
<!DOCTYPE html>
<html>
<head><title>Some Text</title></head>
<body><table>
<tr><td>Character</td><td>Speech</td></tr>
<tr><td>The multitude</td><td>The messiah! Show us the messiah!</td></tr>
<tr><td>Brians mother</td><td>&lt;angry&gt;Now you listen here! He's not the messiah; he's a very
naughty boy! Now go away!&lt;/angry&gt;</td></tr>
<tr><td>The multitude</td><td>Who are you?</td></tr>
<tr><td>Brians mother</td><td>I'm his mother; that's who!</td></tr>
<tr><td>The multitude</td><td>Behold his mother! Behold his mother!</td></tr>
</table></body></html>
```

Cuban primes

```
func cuban_primes(n) {
    1..Inf -> lazy.map {|k| 3*k*(k+1) + 1 }\
                  .grep{ .is_prime }\
                  .first(n)
}

cuban_primes(200).slices(10).each {
    say .map { "%9s" % .commify }.join(' ')
}

say ("\n100,000th cuban prime is: ", cuban_primes(1e5).last.commify)
```

Output:

```
        7        19        37        61       127       271       331       397       547       631
      919     1,657     1,801     1,951     2,269     2,437     2,791     3,169     3,571     4,219
    4,447     5,167     5,419     6,211     7,057     7,351     8,269     9,241    10,267    11,719
   12,097    13,267    13,669    16,651    19,441    19,927    22,447    23,497    24,571    25,117
   26,227    27,361    33,391    35,317    42,841    45,757    47,251    49,537    50,311    55,897
   59,221    60,919    65,269    70,687    73,477    74,419    75,367    81,181    82,171    87,211
   88,237    89,269    92,401    96,661   102,121   103,231   104,347   110,017   112,327   114,661
  115,837   126,691   129,169   131,671   135,469   140,617   144,541   145,861   151,201   155,269
  163,567   169,219   170,647   176,419   180,811   189,757   200,467   202,021   213,067   231,019
  234,361   241,117   246,247   251,431   260,191   263,737   267,307   276,337   279,991   283,669
  285,517   292,969   296,731   298,621   310,087   329,677   333,667   337,681   347,821   351,919
  360,187   368,551   372,769   374,887   377,011   383,419   387,721   398,581   407,377   423,001
  436,627   452,797   459,817   476,407   478,801   493,291   522,919   527,941   553,411   574,219
  584,767   590,077   592,741   595,411   603,457   608,851   611,557   619,711   627,919   650,071
  658,477   666,937   689,761   692,641   698,419   707,131   733,591   742,519   760,537   769,627
  772,669   784,897   791,047   812,761   825,301   837,937   847,477   863,497   879,667   886,177
  895,987   909,151   915,769   925,741   929,077   932,419   939,121   952,597   972,991   976,411
  986,707   990,151   997,057 1,021,417 1,024,921 1,035,469 1,074,607 1,085,407 1,110,817 1,114,471
1,125,469 1,155,061 1,177,507 1,181,269 1,215,397 1,253,887 1,281,187 1,285,111 1,324,681 1,328,671
1,372,957 1,409,731 1,422,097 1,426,231 1,442,827 1,451,161 1,480,519 1,484,737 1,527,247 1,570,357

100,000th cuban prime is: 1,792,617,147,127
```

Cubic special primes

```
func cubic_primes(callback) {

    var prev = 2
    callback(prev)

    loop {
        var curr = (1..Inf -> lazy.map { prev + _**3 }.first { .is_prime })
        callback(curr)
        prev = curr
    }
}

say gather {
    cubic_primes({|k|
        break if (k >= 15000)
        take(k)
    })
}
```

Output:

```
[2, 3, 11, 19, 83, 1811, 2027, 2243, 2251, 2467, 2531, 2539, 3539, 3547, 4547, 5059, 10891, 12619, 1361
```

Cullen and Woodall numbers

```
func cullen(n)  { n * (1 << n) + 1 }
func woodall(n) { n * (1 << n) - 1 }

say "First 20 Cullen numbers:"
say cullen.map(1..20).join(' ')

say "\nFirst 20 Woodall numbers:"
say woodall.map(1..20).join(' ')

say "\nFirst 5 Cullen primes: (in terms of n)"
say 5.by { cullen(_).is_prime }.join(' ')

say "\nFirst 12 Woodall primes: (in terms of n)"
say 12.by { woodall(_).is_prime }.join(' ')
```

Output:

```
First 20 Cullen numbers:
3 9 25 65 161 385 897 2049 4609 10241 22529 49153 106497 229377 491521 1048577 2228225 4718593 9961473

First 20 Woodall numbers:
1 7 23 63 159 383 895 2047 4607 10239 22527 49151 106495 229375 491519 1048575 2228223 4718591 9961471

First 5 Cullen primes: (in terms of n)
1 141 4713 5795 6611

First 12 Woodall primes: (in terms of n)
2 3 6 30 75 81 115 123 249 362 384 462
```

Cumulative standard deviation

Using an object to keep state:

```
class StdDevAccumulator(n=0, sum=0, sumofsquares=0) {
  method <<(num) {
    n += 1
    sum += num
    sumofsquares += num**2
    self
  }

  method stddev {
    sqrt(sumofsquares/n - pow(sum/n, 2))
  }

  method to_s {
    self.stddev.to_s
  }
}

var i = 0
var sd = StdDevAccumulator()
[2,4,4,4,5,5,7,9].each {|n|
    say "adding #{n}: stddev of #{i+=1} samples is #{sd << n}"
}
```

Output:

```
adding 2: stddev of 1 samples is 0
adding 4: stddev of 2 samples is 1
adding 4: stddev of 3 samples is 0.942809041582063365867792482806465385713114583585
adding 4: stddev of 4 samples is 0.866025403784438646763723170752936183471402626905
adding 5: stddev of 5 samples is 0.979795897113271239278913629882356556786378992263
adding 5: stddev of 6 samples is 1
adding 7: stddev of 7 samples is 1.399708424447530341827019471260509366837684274 66
adding 9: stddev of 8 samples is 2
```

Using *static* variables:

```
func stddev(x) {
    static(num=0, sum=0, sum2=0)
    num++
    sqrt(
        (sum2 += x**2) / num -
        (((sum += x) / num)**2)
    )
}

%n(2 4 4 4 5 5 7 9).each { say stddev(_) }
```

Output:

```
0
1
0.9428090415820633658677924828064653857131145835 85
0.8660254037844386467637231707529361834714026269 05
0.9797958971132712392789136298823565567863789922 63
1
1.3997084244475303418270194712605093668376842746 6
2
```

Currency

```
struct Item {
    name, price, quant
}

var check = %q{
    Hamburger   5.50    4000000000000000
    Milkshake   2.86    2
}.lines.grep(/\S/).map { Item(.words...) }

var tax_rate = 0.0765
var fmt = "%-10s %8s %18s %22s\n"

printf(fmt, %w(Item Price Quantity Extension)...)

var subtotal = check.map { |item|
    var extension = Num(item.price)*Num(item.quant)
    printf(fmt, item.name, item.price, item.quant, extension.round(-2))
    extension
}.sum

printf(fmt, '', '', '', '-----------------')
printf(fmt, '', '', 'Subtotal ', subtotal)

var tax = (subtotal * tax_rate -> round(-2))
printf(fmt, '', '', 'Tax ', tax)

var total = subtotal+tax
printf(fmt, '', '', 'Total ', total)
```

Output:

```
Item          Price           Quantity              Extension
Hamburger      5.50   4000000000000000      22000000000000000
Milkshake      2.86                  2                   5.72
                                         -----------------
                             Subtotal    22000000000000005.72
                                  Tax     1683000000000000.44
                                Total    23683000000000006.16
```

Currying

This can be done by using lazy methods:

```
var adder = 1.method(:add)
say adder(3)                #=> 4
```

Or by using a generic curry function:

```
func curry(f, *args1) {
    func (*args2) {
        f(args1..., args2...)
    }
}

func add(a, b) {
    a + b
}

var adder = curry(add, 1)
say adder(3)                #=>4
```

Curzon numbers

```
func is_curzon(n, k) {
    powmod(k, n, k*n + 1).is_congruent(-1, k*n + 1) && (n > 0)
}

for k in (2 .. 10 `by` 2) {
    say "\nFirst 50 Curzon numbers using a base of #{k}:"
    say 50.by {|n| is_curzon(n, k) }.join(' ')
    say ("1000th term: ", 1000.th {|n| is_curzon(n,k) })
}
```

Output:

```
First 50 Curzon numbers using a base of 2:
1 2 5 6 9 14 18 21 26 29 30 33 41 50 53 54 65 69 74 78 81 86 89 90 98 105 113 114 125 134 138 141 146 1
1000th term: 8646

First 50 Curzon numbers using a base of 4:
1 3 7 9 13 15 25 27 37 39 43 45 49 57 67 69 73 79 87 93 97 99 105 115 127 135 139 153 163 165 169 175 1
1000th term: 9375

First 50 Curzon numbers using a base of 6:
1 6 30 58 70 73 90 101 105 121 125 146 153 166 170 181 182 185 210 233 241 242 266 282 290 322 373 381
1000th term: 20717

First 50 Curzon numbers using a base of 8:
1 14 35 44 72 74 77 129 131 137 144 149 150 185 200 219 236 266 284 285 299 309 336 357 381 386 390 392
1000th term: 22176

First 50 Curzon numbers using a base of 10:
1 9 10 25 106 145 190 193 238 253 306 318 349 385 402 462 486 526 610 649 658 678 733 762 810 990 994 1
1000th term: 46845
```

Cycle detection

```
func brent (f, x0) {
    var power = 1
    var λ = 1
    var tortoise = x0
    var hare = f(x0)

    while (tortoise != hare) {
        if (power == λ) {
            tortoise = hare
            power *= 2
            λ = 0
        }
        hare = f(hare)
        λ += 1
    }

    var μ = 0
    tortoise = x0
    hare = x0
    { hare = f(hare) } * λ

    while (tortoise != hare) {
        tortoise = f(tortoise)
        hare = f(hare)
        μ += 1
    }

    return (λ, μ)
}

func cyclical_function(x) { (x*x + 1) % 255 }

var (l, s) = brent(cyclical_function, 3)

var seq = gather {
    var x = 3
    { take(x); x = cyclical_function(x) } * 20
}

say seq.join(', ')+', ...'

say "Cycle length #{l}.";
say "Cycle start index #{s}."
say [seq[s .. (s + l - 1)]]
```

Output:

```
3, 10, 101, 2, 5, 26, 167, 95, 101, 2, 5, 26, 167, 95, 101, 2, 5, 26, 167, 95, ...
Cycle length 6.
Cycle start index 2.
[101, 2, 5, 26, 167, 95]
```

Cyclops numbers

```
func cyclops_numbers(base = 10) {
    Enumerator({|callback|

        var digits = @(1 .. base-1)
```

```
        for k in (0 .. Inf `by` 2) {
            digits.variations_with_repetition(k, {|*a|
                a = (a.first(a.len>>1) + [0] + a.last(a.len>>1))
                callback(a.flip.digits2num(base))
            })
        }
    })
}

func palindromic_cyclops_numbers(base = 10) {
    Enumerator({|callback|

        var digits = @(1 .. base-1)

        for k in (0..Inf) {
            digits.variations_with_repetition(k, {|*a|
                a = (a + [0] + a.flip)
                callback(a.flip.digits2num(base))
            })
        }
    })
}

func prime_cyclops(base = 10) {
    var iter = cyclops_numbers(base)
    Enumerator({|callback|
        iter.each {|n|
            callback(n) if n.is_prime
        }
    })
}

func blind_prime_cyclops(base = 10) {
    var iter = prime_cyclops(base)
    Enumerator({|callback|
        iter.each {|n|
            var k = (n.len(base)-1)>>1
            var r = ipow(base, k)
            if (r*idiv(n, r*base) + n%r -> is_prime) {
                callback(n)
            }
        }
    })
}

func palindromic_prime_cyclops(base = 10) {
    var iter = palindromic_cyclops_numbers(base)
    Enumerator({|callback|
        iter.each {|n|
            callback(n) if n.is_prime
        }
    })
}

for text,f in ([
    ['', cyclops_numbers],
    ['prime', prime_cyclops],
    ['blind prime', blind_prime_cyclops],
    ['palindromic prime', palindromic_prime_cyclops],
]) {

    with (50) {|k|
        say "First #{k} #{text} cyclops numbers:"
        f().first(k).each_slice(10, {|*a|
            a.map { '%7s' % _ }.join(' ').say
        })
```

```
    }

    var min = 10_000_000
    var iter = f()
    var index = 0
    var arr = Enumerator({|callback|
        iter.each {|n|
            callback([index, n]) if (n > min)
            ++index
        }
    }).first(1)[0]
    say "\nFirst #{text} term > #{min.commify}: #{arr[1].commify} at (zero based) index: #
{arr[0].commify}\n"
}
```

Output:

```
First 50  cyclops numbers:
      0     101     102     103     104     105     106     107     108     109
    201     202     203     204     205     206     207     208     209     301
    302     303     304     305     306     307     308     309     401     402
    403     404     405     406     407     408     409     501     502     503
    504     505     506     507     508     509     601     602     603     604

First  term > 10,000,000: 111,101,111 at (zero based) index: 538,084

First 50 prime cyclops numbers:
    101     103     107     109     307     401     409     503     509     601
    607     701     709     809     907   11027   11047   11057   11059   11069
  11071   11083   11087   11093   12011   12037   12041   12043   12049   12071
  12073   12097   13033   13037   13043   13049   13063   13093   13099   14011
  14029   14033   14051   14057   14071   14081   14083   14087   15013   15017

First prime term > 10,000,000: 111,101,129 at (zero based) index: 39,319

First 50 blind prime cyclops numbers:
    101     103     107     109     307     401     503     509     601     607
    701     709     809     907   11071   11087   11093   12037   12049   12097
  13099   14029   14033   14051   14071   14081   14083   14087   15031   15053
  15083   16057   16063   16067   16069   16097   17021   17033   17041   17047
  17053   17077   18047   18061   18077   18089   19013   19031   19051   19073

First blind prime term > 10,000,000: 111,101,161 at (zero based) index: 11,393

First 50 palindromic prime cyclops numbers:
    101   16061   31013   35053   38083   73037   74047   91019   94049 1120211
1150511 1160611 1180811 1190911 1250521 1280821 1360631 1390931 1490941 1520251
1550551 1580851 1630361 1640461 1660661 1670761 1730371 1820281 1880881 1930391
1970791 3140413 3160613 3260623 3310133 3380833 3460643 3470743 3590953 3670763
3680863 3970793 7190917 7250527 7310137 7540457 7630367 7690967 7750577 7820287

First palindromic prime term > 10,000,000: 114,808,411 at (zero based) index: 66
```

Cyclotomic polynomial

Solution based on polynomial interpolation (slow).

```
var Poly = require('Math::Polynomial')
Poly.string_config(Hash(fold_sign => true, prefix => "", suffix => ""))

func poly_interpolation(v) {
    v.len.of {|n| v.len.of {|k| n**k } }.msolve(v)
}

say "First 30 cyclotomic polynomials:"
for k in (1..30) {
    var a = (k+1).of { cyclotomic(k, _) }
    var Φ = poly_interpolation(a)
    say ("Φ(#{k}) = ", Poly.new(Φ...))
}

say "\nSmallest cyclotomic polynomial with n or -n as a coefficient:"
for n in (1..10) {  # very slow
    var k = (1..Inf -> first {|k|
        poly_interpolation((k+1).of { cyclotomic(k, _) }).first { .abs == n }
    })
    say "Φ(#{k}) has coefficient with magnitude #{n}"
}
```

Slightly faster solution, using the **Math::Polynomial::Cyclotomic** Perl module.

```
var Poly = require('Math::Polynomial')
           require('Math::Polynomial::Cyclotomic')

Poly.string_config(Hash(fold_sign => true, prefix => "", suffix => ""))

say "First 30 cyclotomic polynomials:"
for k in (1..30) {
    say ("Φ(#{k}) = ", Poly.new.cyclotomic(k))
}

say "\nSmallest cyclotomic polynomial with n or -n as a coefficient:"
for n in (1..10) {
    var p = Poly.new
    var k = (1..Inf -> first {|k|
        [p.cyclotomic(k).coeff].first { .abs == n }
    })
    say "Φ(#{k}) has coefficient with magnitude = #{n}"
}
```

Output:

```
First 30 cyclotomic polynomials:
Φ(1) = x - 1
Φ(2) = x + 1
Φ(3) = x^2 + x + 1
Φ(4) = x^2 + 1
Φ(5) = x^4 + x^3 + x^2 + x + 1
Φ(6) = x^2 - x + 1
Φ(7) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1
Φ(8) = x^4 + 1
Φ(9) = x^6 + x^3 + 1
Φ(10) = x^4 - x^3 + x^2 - x + 1
Φ(11) = x^10 + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1
Φ(12) = x^4 - x^2 + 1
Φ(13) = x^12 + x^11 + x^10 + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1
Φ(14) = x^6 - x^5 + x^4 - x^3 + x^2 - x + 1
Φ(15) = x^8 - x^7 + x^5 - x^4 + x^3 - x + 1
Φ(16) = x^8 + 1
Φ(17) = x^16 + x^15 + x^14 + x^13 + x^12 + x^11 + x^10 + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2
Φ(18) = x^6 - x^3 + 1
Φ(19) = x^18 + x^17 + x^16 + x^15 + x^14 + x^13 + x^12 + x^11 + x^10 + x^9 + x^8 + x^7 + x^6 + x^5 + x^
Φ(20) = x^8 - x^6 + x^4 - x^2 + 1
Φ(21) = x^12 - x^11 + x^9 - x^8 + x^6 - x^4 + x^3 - x + 1
Φ(22) = x^10 - x^9 + x^8 - x^7 + x^6 - x^5 + x^4 - x^3 + x^2 - x + 1
Φ(23) = x^22 + x^21 + x^20 + x^19 + x^18 + x^17 + x^16 + x^15 + x^14 + x^13 + x^12 + x^11 + x^10 + x^9
Φ(24) = x^8 - x^4 + 1
Φ(25) = x^20 + x^15 + x^10 + x^5 + 1
Φ(26) = x^12 - x^11 + x^10 - x^9 + x^8 - x^7 + x^6 - x^5 + x^4 - x^3 + x^2 - x + 1
Φ(27) = x^18 + x^9 + 1
Φ(28) = x^12 - x^10 + x^8 - x^6 + x^4 - x^2 + 1
Φ(29) = x^28 + x^27 + x^26 + x^25 + x^24 + x^23 + x^22 + x^21 + x^20 + x^19 + x^18 + x^17 + x^16 + x^15
Φ(30) = x^8 + x^7 - x^5 - x^4 - x^3 + x + 1

Smallest cyclotomic polynomial with n or -n as a coefficient:
Φ(1) has coefficient with magnitude = 1
Φ(105) has coefficient with magnitude = 2
Φ(385) has coefficient with magnitude = 3
Φ(1365) has coefficient with magnitude = 4
Φ(1785) has coefficient with magnitude = 5
Φ(2805) has coefficient with magnitude = 6
Φ(3135) has coefficient with magnitude = 7
^C
```

Damm algorithm

```
func damm(digits) {
    static tbl = [
          [0, 3, 1, 7, 5, 9, 8, 6, 4, 2],
          [7, 0, 9, 2, 1, 5, 4, 8, 6, 3],
          [4, 2, 0, 6, 8, 7, 1, 3, 5, 9],
          [1, 7, 5, 0, 9, 8, 3, 4, 2, 6],
          [6, 1, 2, 3, 0, 4, 5, 9, 7, 8],
          [3, 6, 7, 4, 2, 0, 9, 5, 8, 1],
          [5, 8, 6, 9, 7, 2, 0, 1, 3, 4],
          [8, 9, 4, 5, 3, 6, 2, 0, 1, 7],
          [9, 4, 3, 8, 6, 1, 7, 2, 0, 5],
          [2, 5, 8, 1, 4, 3, 6, 7, 9, 0],
    ]

    !digits.flip.reduce({|row,col| tbl[row][col] }, 0)
}

for n in [5724, 5727, 112946] {
    say "#{n}:\tChecksum digit #{ damm(n.digits) ? '' : 'in'}correct."
}
```

Output:

```
5724:   Checksum digit correct.
5727:   Checksum digit incorrect.
112946: Checksum digit correct.
```

Date format

```
var time = Time.local
say time.ctime
say time.strftime("%Y-%m-%d")
say time.strftime("%A, %B %d, %Y")
```

Output:

```
Fri Oct 17 12:57:02 2014
2014-10-17
Friday, October 17, 2014
```

Date manipulation

```
var dt = frequire('DateTime::Format::Strptime')

var input =  'March 7 2009 7:30pm EST'
input.sub!('EST', 'America/New_York')

say dt.strptime('%b %d %Y %I:%M%p %O', input)   \
      .add(hours => 12)                         \
      .set_time_zone('America/Edmonton')        \
      .format_cldr('MMMM d yyyy h:mma zzz')
```

Output:

```
March 8 2009 6:30AM MDT
```

Day of the week

```
require('Time::Local')

for year in (2008 .. 2121) {
    var time = %S<Time::Local>.timelocal(0,0,0,25,11,year)
    var wd = Time(time).local.wday
    if (wd == 0) {
        say "25 Dec #{year} is Sunday"
    }
}
```

Output:

```
25 Dec 2011 is Sunday
25 Dec 2016 is Sunday
25 Dec 2022 is Sunday
25 Dec 2033 is Sunday
25 Dec 2039 is Sunday
25 Dec 2044 is Sunday
25 Dec 2050 is Sunday
25 Dec 2061 is Sunday
25 Dec 2067 is Sunday
25 Dec 2072 is Sunday
25 Dec 2078 is Sunday
25 Dec 2089 is Sunday
25 Dec 2095 is Sunday
25 Dec 2101 is Sunday
25 Dec 2107 is Sunday
25 Dec 2112 is Sunday
25 Dec 2118 is Sunday
```

Days between dates

```
require('Date::Calc')

func days_diff(a,b) {
    %S<Date::Calc>.Delta_Days(a.split('-')..., b.split('-')...)
}

var date1 = "1970-01-01"
var date2 = "2019-10-02"

say "Date 1: #{date1}"
say "Date 2: #{date2}"

var days = days_diff(date1, date2)

say "There are #{days} days between these dates"
```

Output:

```
Date 1: 1970-01-01
Date 2: 2019-10-02
There are 18171 days between these dates
```

Death Star

Writes a PGM to stdout.

```
func sq(*nums) {
    nums »**» 2 «+»;
}

func hitf(sph, x, y) {
    x -= sph[0];
    y -= sph[1];

    var z = (sq(sph[3]) - sq(x, y));
    z < 0 && return();

    z.sqrt!;
    [sph[2] - z, sph[2] + z];
}

func normalize(v) {
    var n = sq(v...).sqrt;
    v »/» n;
}

func dot(x, y) {
    var s = (x[0]*y[0] + x[1]*y[1] + x[2]*y[2]);
    s > 0 ? s : 0;
}

var pos = [120, 120, 0, 120];
var neg = [-77, -33, -100, 190];
var light = normalize([-12, 13, -10]);

func draw(k, amb) {
    STDOUT.binmode(':raw');
    print ("P5\n", pos[0]*2 + 3, " ", pos[1]*2 + 3, "\n255\n");

    for y in ((pos[1] - pos[3] - 1) .. (pos[1] + pos[3] + 1)) {
        var row = [];
        for x in ((pos[0] - pos[3] - 1) .. (pos[0] + pos[3] + 1)) {
            var hit = 0;
            var hs = [];
            var h = hitf(pos, x, y);

            if    (!h)                      { hit = 0; h  = [0, 0] }
            elsif (!(hs = hitf(neg, x, y))) { hit = 1; hs = [0, 0] }
            elsif (hs[0] > h[0])            { hit = 1 }
            elsif (hs[1] > h[0])            { hit = (hs[1] > h[1] ? 0 : 2) }
            else                            { hit = 1 };

            var (val, v);
            given(hit) {
                when (0) { val = 0}
                when (1) { v = [x-pos[0], y-pos[1], h[0]-pos[2]] }
                default  { v = [neg[0]-x, neg[1]-y, neg[2]-hs[1]] }
            }
```

```
            if (v) {
                v = normalize(v);
                val = int((dot(v, light)**k + amb) * 255);
                val = (val > 255 ? 255 : (val < 0 ? 0 : val));
            };
            row.append(val);
        }
        print 'C*'.pack(row...);
    }
}

draw(2, 0.2);
```

Deceptive numbers

```
say 100.by {|n|
    n.is_composite && (divmod(powmod(10, n-1, n)-1, 9, n) == 0)
}.join(' ')
```

Output:

```
91 259 451 481 703 1729 2821 2981 3367 4141 4187 5461 6533 6541 6601 7471 7777 8149 8401 8911 10001 111
```

Decimal floating point number to binary

```
func dec2bin(String n) {
    Num(Num(n, 10).base(2), 10)
}

func bin2dec(String n) {
    Num(Num(n, 10).base(10), 2)
}

with("23.34375")   { |s| say ("  #{s} => ", dec2bin(s)) }
with("1011.11101") { |s| say (  "#{s} => ", bin2dec(s)) }
```

Output:

```
  23.34375 => 10111.01011
1011.11101 => 11.90625
```

Decision tables

```
func decide (q, s) {

    var bits = q.map { |p|
        read("#{p.value}? ", String) ~~ /^y/i ? 1 : 0
    }

    var n = with (0) { |t|
        bits.each { |b|
            t <<= 1
            t |= b
        }
        1 << t
    }

    s.grep { .key & n }.map{ .value }.each { |ans|
        say "   #{ans}"
    }
}

loop {
    decide(
      [
        Pair("Y Y Y Y N N N N", "Printer does not print"),
        Pair("Y Y N N Y Y N N", "A red light is flashing"),
        Pair("Y N Y N Y N Y N", "Printer is unrecognised"),
      ],
      [
        Pair(0b0_0_1_0_0_0_0_0, "Check the power cable"),
        Pair(0b1_0_1_0_0_0_0_0, "Check the printer-computer cable"),
        Pair(0b1_0_1_0_1_0_1_0, "Ensure printer software is installed"),
        Pair(0b1_1_0_0_1_1_0_0, "Check/replace ink"),
        Pair(0b0_1_0_1_0_0_0_0, "Check for paper jam"),
      ]
    )
    say ''
}
```

Output:

```
Printer does not print? y
A red light is flashing? n
Printer is unrecognised? n
   Check for paper jam

Printer does not print? n
A red light is flashing? n
Printer is unrecognised? y
   Ensure printer software is installed

Printer does not print? y
A red light is flashing? y
Printer is unrecognised? n
   Check/replace ink
   Check for paper jam

Printer does not print? n
A red light is flashing? y
Printer is unrecognised? y
   Ensure printer software is installed
   Check/replace ink

Printer does not print? ^C
```

Deepcopy

Object.dclone() returns a deep clone of any mutable object.

```
var src = Hash(foo => 0, bar => [0,1])

# Add a cyclic reference
src{:baz} = src

# Make a deep clone
var dst = src.dclone

# The address of src
say src.object_id
say src{:baz}.object_id

# The address of dst
say dst.object_id
say dst{:baz}.object_id
```

Output:

```
15154128
15154128
25296304
25296304
```

Define a primitive data type

```
subset Integer    < Number  { .is_int }
subset MyIntLimit < Integer { . ~~ (1 .. 10) }

class MyInt(value < MyIntLimit) {

    method to_s      { value.to_s }
    method get_value { value.get_value }

    method ==(Number x) { value == x }
    method ==(MyInt  x) { value == x.value }

    method AUTOLOAD(_, name, *args) {
        var results = [value.(name)(args.map {|n| Number(n) }...)]
        results.map{|r| r.kind_of(Number) ? MyInt(r.int) : r}...
    }
}

#
## Example:
#
var a = MyInt(2)    # creates a new object of type `MyInt`
a += 7              # adds 7 to a
say a               # => 9
say a/2             # => 4

var b = (a - 3)     # b is of type `MyInt`
say b               # => 6

say a.as_hex.dump   # => "9" -- an hexadecimal string

a -= 6              # a=3
var c = (a + b)     # MyInt(3) + MyInt(6)
say c               # => 9
say c.class         # => MyInt

a *= 2              # a=6
say a+b             # error: class `MyInt` does not match MyInt(12)
```

Delegates

```
class NonDelegate { }

class Delegate {
    method thing {
        return "delegate implementation"
    }
}

class Delegator (delegate = null) {
    method operation {

        if (delegate.respond_to(:thing)) {
            return delegate.thing
        }

        return "default implementation"
    }
}

var d = Delegator()
say "empty: #{d.operation}"
d.delegate = NonDelegate()
say "NonDelegate: #{d.operation}"
d.delegate = Delegate()
say "Delegate: #{d.operation}"
```

Output:

```
empty: default implementation
NonDelegate: default implementation
Delegate: delegate implementation
```

Delete a file

```
# here
%f'input.txt' -> delete
%d'docs'      -> delete

# root dir
Dir.root + %f'input.txt' -> delete
Dir.root + %d'docs'      -> delete
```

Deming's funnel

```
func x̄(a) {
    a.sum / a.len
}

func σ(a) {
    sqrt(x̄(a.map{.**2}) - x̄(a)**2)
}

const Δ = (%n<
    -0.533  0.270  0.859 -0.043 -0.205 -0.127 -0.071  0.275
     1.251 -0.231 -0.401  0.269  0.491  0.951  1.150  0.001
    -0.382  0.161  0.915  2.080 -2.337  0.034 -0.126  0.014
     0.709  0.129 -1.093 -0.483 -1.193  0.020 -0.051  0.047
    -0.095  0.695  0.340 -0.182  0.287  0.213 -0.423 -0.021
    -0.134  1.798  0.021 -1.099 -0.361  1.636 -1.134  1.315
     0.201  0.034  0.097 -0.170  0.054 -0.553 -0.024 -0.181
    -0.700 -0.361 -0.789  0.279 -0.174 -0.009 -0.323 -0.658
     0.348 -0.528  0.881  0.021 -0.853  0.157  0.648  1.774
    -1.043  0.051  0.021  0.247 -0.310  0.171  0.000  0.106
     0.024 -0.386  0.962  0.765 -0.125 -0.289  0.521  0.017
     0.281 -0.749 -0.149 -2.436 -0.909  0.394 -0.113 -0.598
     0.443 -0.521 -0.799  0.087
> ~Z+ %n<
     0.136  0.717  0.459 -0.225  1.392  0.385  0.121 -0.395
     0.490 -0.682 -0.065  0.242 -0.288  0.658  0.459  0.000
     0.426  0.205 -0.765 -2.188 -0.742 -0.010  0.089  0.208
     0.585  0.633 -0.444 -0.351 -1.087  0.199  0.701  0.096
    -0.025 -0.868  1.051  0.157  0.216  0.162  0.249 -0.007
     0.009  0.508 -0.790  0.723  0.881 -0.508  0.393 -0.226
     0.710  0.038 -0.217  0.831  0.480  0.407  0.447 -0.295
     1.126  0.380  0.549 -0.445 -0.046  0.428 -0.074  0.217
    -0.822  0.491  1.347 -0.141  1.230 -0.044  0.079  0.219
     0.698  0.275  0.056  0.031  0.421  0.064  0.721  0.104
    -0.729  0.650 -1.103  0.154 -1.720  0.051 -0.385  0.477
     1.537 -0.901  0.939 -0.411  0.341 -0.411  0.106  0.224
    -0.947 -1.424 -0.542 -1.032
>.map{ .i })

const rules = [
    { 0 },
    {|_,dz| -dz },
    {|z,dz| -z - dz },
    {|z,dz| z + dz },
]

for i,v in (rules.kv) {
    say "Rule #{i+1}:"
    var target = 0
    var z = gather {
        Δ.each { |d|
            take(target + d)
            target = v.run(target, d)
        }
    }
    printf("Mean    x, y   : %.4f %.4f\n", x̄(z.map{.re}), x̄(z.map{.im}))
    printf("Std dev x, y   : %.4f %.4f\n", σ(z.map{.re}), σ(z.map{.im}))
}
```

Output:

```
Rule 1:
Mean    x, y   : 0.0004 0.0702
Std dev x, y   : 0.7153 0.6462
Rule 2:
Mean    x, y   : 0.0009 -0.0103
Std dev x, y   : 1.0371 0.8999
Rule 3:
Mean    x, y   : 0.0439 -0.0063
Std dev x, y   : 7.9871 4.7784
Rule 4:
Mean    x, y   : 3.1341 5.4210
Std dev x, y   : 1.5874 3.9304
```

Department numbers

```
@(1..7)->combinations(3, {|*a|
    a.sum == 12 || next
    a.permutations {|*b|
        b[0].is_even || next
        say (%w(police fire sanitation) ~Z b -> join(" "))
    }
})
```

Output:

```
["police", 4] ["fire", 1] ["sanitation", 7]
["police", 4] ["fire", 7] ["sanitation", 1]
["police", 6] ["fire", 1] ["sanitation", 5]
["police", 6] ["fire", 5] ["sanitation", 1]
["police", 2] ["fire", 3] ["sanitation", 7]
["police", 2] ["fire", 7] ["sanitation", 3]
["police", 2] ["fire", 4] ["sanitation", 6]
["police", 2] ["fire", 6] ["sanitation", 4]
["police", 4] ["fire", 2] ["sanitation", 6]
["police", 4] ["fire", 6] ["sanitation", 2]
["police", 6] ["fire", 2] ["sanitation", 4]
["police", 6] ["fire", 4] ["sanitation", 2]
["police", 4] ["fire", 3] ["sanitation", 5]
["police", 4] ["fire", 5] ["sanitation", 3]
```

Descending primes

```
func primes_with_descending_digits(base = 10) {

    var list = []
    var digits = @(1..^base)

    var end_digits = digits.grep { .is_coprime(base) }
    list << digits.grep { .is_prime && !.is_coprime(base) }...

    for k in (0 .. digits.end) {
        digits.combinations(k, {|*a|
            var v = a.digits2num(base)
            end_digits.each {|d|
                var n = (v*base + d)
                next if ((n >= base) && (a[0] <= d))
                list << n if n.is_prime
            }
        })
    }

    list.sort
}

var base = 10
var arr = primes_with_descending_digits(base)

say "There are #{arr.len} descending primes in base #{base}.\n"

arr.each_slice(8, {|*a|
    say a.map { '%9s' % _ }.join(' ')
})
```

Output:

```
There are 87 descending primes in base 10.

        2         3         5         7        31        41        43        53
       61        71        73        83        97       421       431       521
      541       631       641       643       653       743       751       761
      821       853       863       941       953       971       983      5431
     6421      6521      7321      7541      7621      7643      8431      8521
     8543      8641      8731      8741      8753      8761      9421      9431
     9521      9631      9643      9721      9743      9851      9871     75431
    76421     76541     76543     86531     87421     87541     87631     87641
    87643     94321     96431     97651     98321     98543     98621     98641
    98731    764321    865321    876431    975421    986543    987541    987631
  8764321   8765321   9754321   9875321  97654321  98764321  98765431
```

Detect division by zero

The numerical system of Sidef evaluates `x/0` to `+/-Inf`.

```
func div_check(a, b){
    var result = a/b
    result.abs == Inf ? nil : result
}

say div_check(10, 2)  # 5
say div_check(1, 0)   # nil (detected)
```

Alternatively, we can do:

```
func div_check(a, b){
    Perl.eval("#{a} / #{b}")
}

say div_check(10, 2)  # 5
say div_check(1, 0)   # nil (detected)
```

Determinant and permanent

The `determinant` method is provided by the Array class.

```
class Array {
    method permanent {
        var r = @^self.len

        var sum = 0
        r.permutations { |*a|
            var prod = 1
            [a,r].zip {|row,col| prod *= self[row][col] }
            sum += prod
        }

        return sum
    }
}

var m1 = [[1,2],[3,4]]

var m2 = [[1, 2, 3, 4],
          [4, 5, 6, 7],
          [7, 8, 9, 10],
          [10, 11, 12, 13]]

var m3 = [[0, 1, 2, 3, 4],
          [5, 6, 7, 8, 9],
          [10, 11, 12, 13, 14],
          [15, 16, 17, 18, 19],
          [20, 21, 22, 23, 24]]

[m1, m2, m3].each { |m|
  say "determinant:\t #{m.determinant}\npermanent:\t #{m.permanent}\n"
}
```

Output:

```
determinant:     -2
permanent:       10

determinant:     0
permanent:       29556

determinant:     0
permanent:       6778800
```

Determine if a string has all the same characters

```
func analyze_string(str) {
    var chars = str.chars
    chars.range.to_a.each_cons(2, {|a,b|
        chars[a] == chars[b] || return b
    })
    return str.len
}

var strings = ["", "   ", "2", "333", ".55", "tttTTT", "4444   444k", "pépé",
"                                                                      ",
"                                                                      "]

strings.each {|str|
    print "'#{str}' (size #{str.len}): "
    var idx = analyze_string(str)

    if (idx == str.len) {
        say "all the same."
    }
    else {
        say "first different char '#{str[idx]}' (#{'%#x' % str[idx].ord}) at position #{idx}."
    }
}
```

Output:

```
'' (size 0): all the same.
'   ' (size 3): all the same.
'2' (size 1): all the same.
'333' (size 3): all the same.
'.55' (size 3): first different char '5' (0x35) at position 1.
'tttTTT' (size 6): first different char 'T' (0x54) at position 3.
'4444   444k' (size 11): first different char ' ' (0x20) at position 4.
'pépé' (size 4): first different char 'é' (0xe9) at position 1.
'                                                                      ' (size 4): first different char '
'                                                                      ' (size 4): all the same.
```

Determine if a string has all unique characters

```
func index_duplicates(str) {
    gather {
        for k,v in (str.chars.kv) {
            var i = str.index(v, k+1)
            take([k, i]) if (i != -1)
        }
    }
}

var strings = [
    "", ".", "abcABC", "XYZ ZYX",
    "1234567890ABCDEFGHIJKLMN0PQRSTUVWXYZ",
    "01234567890ABCDEFGHIJKLMN0PQRSTUVWXYZ0X",
    "hétérogénéité", "                                                                ",
"😊😀                                  😊                ",

"                🐟

]

strings.each {|str|
    print "\n'#{str}' (size: #{str.len}) "
    var dups = index_duplicates(str)
    say "has duplicated characters:" if dups
    for i,j in (dups) {
        say "#{str[i]} (#{'%#x' % str[i].ord}) in positions: #{i}, #{j}"
    }
    say "has no duplicates." if !dups
}
```

Output:

```
'' (size: 0) has no duplicates.

'.' (size: 1) has no duplicates.

'abcABC' (size: 6) has no duplicates.

'XYZ ZYX' (size: 7) has duplicated characters:
X (0x58) in positions: 0, 6
Y (0x59) in positions: 1, 5
Z (0x5a) in positions: 2, 4

'1234567890ABCDEFGHIJKLMN0PQRSTUVWXYZ' (size: 36) has duplicated characters:
0 (0x30) in positions: 9, 24

'01234567890ABCDEFGHIJKLMN0PQRSTUVWXYZ0X' (size: 39) has duplicated characters:
0 (0x30) in positions: 0, 10
0 (0x30) in positions: 10, 25
0 (0x30) in positions: 25, 37
X (0x58) in positions: 34, 38

'hétérogénéité' (size: 13) has duplicated characters:
é (0xe9) in positions: 1, 3
t (0x74) in positions: 2, 11
é (0xe9) in positions: 3, 7
é (0xe9) in positions: 7, 9
é (0xe9) in positions: 9, 12

'                                                              ' (size: 4) has no duplicates.

'😍😀                                  😍                ' (size: 6) has duplicated characters:
😍 (0x1f60d) in positions: 0, 4
                (0x1f64c) in positions: 2, 5

'               🐟
                (0x1f421) in positions: 2, 7
```

Determine if a string is collapsible

```
func squeeze(str) {
    str.gsub(/(.)\1+/, {|s1| s1 })
}

var strings = ["",
        '"If I were two-faced, would I be wearing this one?" --- Abraham Lincoln ',
        "..111111111111111111111111111111111111111111111111111111111111117777888",
        "I never give 'em hell, I just tell the truth, and they think it's hell. ",
        "                                                  ---  Harry S Truman  ",
        "The better the 4-wheel drive, the further you'll be from help when ya get stuck!",
        "headmistressship",
        "aardvark",
        "😍😀                                  😍😍😍               "]

strings.each {|str|
    var ssq = squeeze(str)
    say "«««#{str}»»» (length: #{str.len})"
    say "«««#{ssq}»»» (length: #{ssq.len})\n"
}
```

Determine if a string is numeric

There is the the *String.looks_like_number* method, which returns true when a given strings looks like a number:

```
say "0.1E-5".looks_like_number       #=> true
```

Alternatively, we can use regular expressions to determine this:

```
func is_numeric(s) {
    (s ~~ /^[+-]?+(?=\.?[0-9])[0-9_]*+(?:\.[0-9_]++)?(?:[Ee](?:[+-]?+[0-9_]+))?\z/) ||
    (s ~~ /^0(?:b[10_]*|x[0-9A-Fa-f_]*|[0-9_]+\b)\z/)
}
```

Sample:

```
var strings = %w(0 0.0 -123 abc 0x10 0xABC 123a -123e3 0.1E-5 50e);
for str in strings {
    say ("%9s => %s" % (str, is_numeric(str)))
}
```

Output:

```
        0 => true
      0.0 => true
     -123 => true
      abc => false
     0x10 => true
    0xABC => true
     123a => false
   -123e3 => true
   0.1E-5 => true
      50e => false
```

Determine if a string is squeezable

```
func squeeze(str, c) {
    str.gsub(Regex("(" + c.escape + ")" + '\1+'), {|s1| s1 })
}

var strings = ["",
        '"If I were two-faced, would I be wearing this one?" --- Abraham Lincoln ',
        "..111111111111111111111111111111111111111111111111111111111117777888",
        "I never give 'em hell, I just tell the truth, and they think it's hell. ",
        "                                                    ---  Harry S Truman  ",
        "😍😀                                  😍😍😍                 "]

var squeeze_these = ["", "-", "7", ".", " -r", "😍"]

[strings, squeeze_these].zip {|str,st|
    say "    original: «««#{str}»»» (length: #{str.len})"
    st.each {|c|
        var ssq = squeeze(str, c)
        say "'#{c}'-squeezed: ««««#{ssq}»»»» (length: #{ssq.len})"
    }
    say ''
}
```

Determine if only one instance is running

```
# For this to work, you need to explicitly
# store the returned fh inside a variable.
var fh = File(__FILE__).open_r

# Now call the flock() method on it
fh.flock(File.LOCK_EX | File.LOCK_NB) ->
    || die "I'm already running!"

# Your code here...
say "Running..."
Sys.sleep(20)
say 'Done!'
```

Dice game probabilities

```
func combos(sides, n) {
    n || return [1]
    var ret = ([0] * (n*sides.max + 1))
    combos(sides, n-1).each_kv { |i,v|
        v && for s in sides { ret[i + s] += v }
    }
    return ret
}

func winning(sides1, n1, sides2, n2) {
    var (p1, p2) = (combos(sides1, n1), combos(sides2, n2))
    var (win,loss,tie) = (0,0,0)
    p1.each_kv { |i, x|
        win  += x*p2.ft(0,i-1).sum
        tie  += x*p2.ft(i, i).sum
        loss += x*p2.ft(i+1).sum
    }
    [win, tie, loss] »/» p1.sum*p2.sum
}

func display_results(String title, Array res) {
    say "=> #{title}"
    for name, prob in (%w(p₁\ win tie p₂\ win) ~Z res) {
        say "P(#{'%6s' % name}) =~ #{prob.round(-11)} (#{prob.as_frac})"
    }
    print "\n"
}

display_results('9D4 vs 6D6',  winning(range(1, 4), 9, range(1,6), 6))
display_results('5D10 vs 6D7', winning(range(1,10), 5, range(1,7), 6))
```

Output:

```
=> 9D4 vs 6D6
P(p₁ win) =~ 0.57314407678 (48679795/84934656)
P(   tie) =~ 0.07076616984 (144252007/2038431744)
P(p₂ win) =~ 0.35608975338 (725864657/2038431744)

=> 5D10 vs 6D7
P(p₁ win) =~ 0.64278862872 (3781171969/5882450000)
P(   tie) =~ 0.04449603031 (523491347/11764900000)
P(p₂ win) =~ 0.31271534097 (735812943/2352980000)
```

Digit fifth powers

```
func digit_nth_powers(n, base=10) {

    var D = @(^base)
    var P = D.map {|d| d**n }
    var A = []
    var m = (base-1)**n

    for(var (k, t) = (1, 1); k*m >= t; (++k, t*=base)) {
        D.combinations_with_repetition(k, {|*c|
            var v = c.sum {|d| P[d] }
            A.push(v) if (v.digits(base).sort == c)
        })
    }

    A.sort.grep { _ > 1 }
}

for n in (3..8) {
    var a = digit_nth_powers(n)
    say "Sum of #{n}-th powers of their digits: #{a.join(' + ')} = #{a.sum}"
}
```

Output:

```
Sum of 3-th powers of their digits: 153 + 370 + 371 + 407 = 1301
Sum of 4-th powers of their digits: 1634 + 8208 + 9474 = 19316
Sum of 5-th powers of their digits: 4150 + 4151 + 54748 + 92727 + 93084 + 194979 = 443839
Sum of 6-th powers of their digits: 548834 = 548834
Sum of 7-th powers of their digits: 1741725 + 4210818 + 9800817 + 9926315 + 14459929 = 40139604
Sum of 8-th powers of their digits: 24678050 + 24678051 + 88593477 = 137949578
```

Digital root

```
func digroot (r, base = 10) {
    var root = r.base(base)
    var persistence = 0
    while (root.len > 1) {
        root = root.chars.map{|n| Number(n, 36) }.sum.base(base)
        ++persistence
    }
    return(persistence, root)
}

var nums = [5, 627615, 39390, 588225, 393900588225]
var bases = [2, 3, 8, 10, 16, 36]
var fmt = "%25s(%2s): persistance = %s, root = %2s\n"

nums << (550777011503 *
         1055648978939934128133070405387866907180899631804629134066821924 79)

bases.each { |b|
    nums.each { |n|
        var x = n.base(b)
        x = 'BIG' if (x.len > 25)
        fmt.printf(x, b, digroot(n, b))
    }
    print "\n"
}
```

Output:

```
                    101( 2): persistance = 2, root =  1
     100110010011100111111( 2): persistance = 3, root =  1
         10011001110111110( 2): persistance = 3, root =  1
     100011111001110000001( 2): persistance = 3, root =  1
                    BIG( 2): persistance = 3, root =  1
                    BIG( 2): persistance = 4, root =  1

                     12( 3): persistance = 2, root =  1
            1011212221000( 3): persistance = 3, root =  1
               2000000220( 3): persistance = 2, root =  2
            1002212220010( 3): persistance = 3, root =  1
110112220112111001100000( 3): persistance = 3, root =  1
                    BIG( 3): persistance = 4, root =  1

                       5( 8): persistance = 0, root =  5
                 2311637( 8): persistance = 3, root =  2
                  114736( 8): persistance = 3, root =  1
                 2174701( 8): persistance = 3, root =  1
            5566623376301( 8): persistance = 3, root =  4
                    BIG( 8): persistance = 3, root =  3

                       5(10): persistance = 0, root =  5
                  627615(10): persistance = 2, root =  9
                   39390(10): persistance = 2, root =  6
                  588225(10): persistance = 2, root =  3
             393900588225(10): persistance = 2, root =  9
                    BIG(10): persistance = 3, root =  4

                       5(16): persistance = 0, root =  5
                    9939f(16): persistance = 2, root =  f
                     99de(16): persistance = 2, root =  f
                    8f9c1(16): persistance = 2, root =  f
                5bb64dfcc1(16): persistance = 2, root =  f
                    BIG(16): persistance = 3, root =  7

                       5(36): persistance = 0, root =  5
                    dg9r(36): persistance = 2, root =  u
                     ue6(36): persistance = 2, root =  f
                    clvl(36): persistance = 2, root =  f
                50ye8n29(36): persistance = 2, root =  p
                    BIG(36): persistance = 3, root =  h
```

Digital root/Multiplicative digital root

```
func mdroot(n) {
  var (mdr, persist) = (n, 0)
  while (mdr >= 10) {
    mdr = mdr.digits.prod
    ++persist
  }
  [mdr, persist]
}

say "Number: MDR  MP\n======  ===  =="
[123321, 7739, 893, 899998].each{|n| "%6d: %3d %3d\n" \
                           .printf(n, mdroot(n)...) }

var counter = Hash()

Inf.times { |j|
  counter{mdroot(j).first} := [] << j
  break if counter.values.all {|v| v.len >= 5 }
}

say "\nMDR: [n0..n4]\n===  ========"
10.times {|i| "%3d: %s\n".printf(i, counter{i}.first(5)) }
```

Output:

```
Number: MDR  MP
======  ===  ==
123321:   8   3
  7739:   8   3
   893:   2   3
899998:   0   2

MDR: [n0..n4]
===  ========
  0: [0, 10, 20, 25, 30]
  1: [1, 11, 111, 1111, 11111]
  2: [2, 12, 21, 26, 34]
  3: [3, 13, 31, 113, 131]
  4: [4, 14, 22, 27, 39]
  5: [5, 15, 35, 51, 53]
  6: [6, 16, 23, 28, 32]
  7: [7, 17, 71, 117, 171]
  8: [8, 18, 24, 29, 36]
  9: [9, 19, 33, 91, 119]
```

Dijkstra's algorithm

```
class Graph(*args) {

    struct Node {
        String name,
        Array edges = [],
        Number dist = Inf,
        Node prev = nil,
        Bool visited = false,
    }

    struct Edge {
        Number weight,
```

```
        Node vertex,
    }

    has g = Hash()

    method init {
        args.each { |a|
            self.add_edge(a...)
        }
    }

    method get(name) {
        g{name}
    }

    method add_edge(a, b, weight) {
        g{a} ||= Node(name: a)
        g{b} ||= Node(name: b)
        g{a}.edges << Edge(weight, g{b})
    }

    method push_priority(a, v) {
        var i = 0
        var j = a.end
        while (i <= j) {
            var k = ((i + j) // 2)
            if (a[k].dist >= v.dist) {
                j = k-1
            }
            else {
                i = k+1
            }
        }
        a.insert(i, v)
    }

    method dijkstra(a, b) {
        g{a}.dist = 0
        var h = []
        self.push_priority(h, g{a})
        while (!h.is_empty) {
            var v = h.shift
            break if (v.name == b)
            v.visited = true
            v.edges.each { |e|
                var u = e.vertex
                if (!u.visited && (v.dist+e.weight <= u.dist)) {
                    u.prev = v
                    u.dist = (v.dist + e.weight)
                    self.push_priority(h, u)
                }
            }
        }
    }
}

var g = Graph(
    ["a", "b", 7],
    ["a", "c", 9],
    ["a", "f", 14],
    ["b", "c", 10],
    ["b", "d", 15],
    ["c", "d", 11],
    ["c", "f", 2],
    ["d", "e", 6],
    ["e", "f", 9],
```

```
)

g.dijkstra('a', 'e')

var v = g.get('e')
var a = []
while (v != nil) {
    a << v.name
    v = v.prev
}

var path = a.reverse.join
say "#{g.get('e').dist} #{path}"
```

Output:

```
26 acde
```

Dinesman's multiple-dwelling problem

```
func dinesman(problem) {
  var lines = problem.split('.')
  var names = lines.first.scan(/\b[A-Z]\w*/)
  var re_names = Regex(names.join('|'))

  # Later on, search for these keywords (the word "not" is handled separately).
  var words = %w(first second third fourth fifth sixth seventh eighth ninth tenth
                 bottom top higher lower adjacent)
  var re_keywords = Regex(words.join('|'))

  # Build an array of lambda's
  var predicates = lines.ft(1, lines.end-1).map{ |line|
    var keywords = line.scan(re_keywords)
    var (name1, name2) = line.scan(re_names)...

    keywords.map{ |keyword|
      var l = do {
        given(keyword) {
            when ("bottom")   { ->(c) { c.first == name1 } }
            when ("top")      { ->(c) { c.last == name1 } }
            when ("higher")   { ->(c) { c.index(name1) > c.index(name2) } }
            when ("lower")    { ->(c) { c.index(name1) < c.index(name2) } }
            when ("adjacent") { ->(c) { c.index(name1) - c.index(name2) -> abs == 1 } }
            default           { ->(c) { c[words.index(keyword)] == name1 } }
        }
      }
      line ~~ /\bnot\b/ ? func(c) { l(c) -> not } : l;  # handle "not"
    }
  }.flat

  names.permutations { |*candidate|
    predicates.all { |predicate| predicate(candidate) } && return candidate
  }
}
```

Function invocation:

```
var demo1 = "Abe Ben Charlie David. Abe not second top. not adjacent Ben Charlie.
David Abe adjacent. David adjacent Ben. Last line."

var demo2 = "A B C D. A not adjacent D. not B adjacent higher C. C lower D. Last line"

var problem1 = "Baker, Cooper, Fletcher, Miller, and Smith live on different floors of an apartment
house that
contains only five floors. Baker does not live on the top floor. Cooper does not live on the bottom
floor.
Fletcher does not live on either the top or the bottom floor. Miller lives on a higher floor than
does Cooper.
Smith does not live on a floor adjacent to Fletcher's. Fletcher does not live on a floor adjacent to
Cooper's.
Where does everyone live?"

var problem2 = "Baker, Cooper, Fletcher, Miller, Guinan, and Smith
live on different floors of an apartment house that contains
only seven floors. Guinan does not live on either the top or the third or the fourth floor.
Baker does not live on the top floor. Cooper
does not live on the bottom floor. Fletcher does not live on
either the top or the bottom floor. Miller lives on a higher
floor than does Cooper. Smith does not live on a floor
adjacent to Fletcher's. Fletcher does not live on a floor
adjacent to Cooper's. Where does everyone live?"

[demo1, demo2, problem1, problem2].each{|problem| say dinesman(problem).join("\n"); say '' }
```

Output:

```
Ben
David
Abe
Charlie

B
A
C
D

Smith
Cooper
Baker
Fletcher
Miller

Baker
Cooper
Miller
Fletcher
Guinan
Smith
```

```
var names = %w(Baker Cooper Fletcher Miller Smith)

var predicates = [
    ->(c){ :Baker != c.last },
    ->(c){ :Cooper != c.first },
    ->(c){ (:Fletcher != c.first) && (:Fletcher != c.last) },
    ->(c){ c.index(:Miller) > c.index(:Cooper) },
    ->(c){ (c.index(:Smith) - c.index(:Fletcher)).abs != 1 },
    ->(c){ (c.index(:Cooper) - c.index(:Fletcher)).abs != 1 },
]

names.permutations { |*candidate|
    if (predicates.all {|predicate| predicate(candidate) }) {
        say candidate.join("\n")
        break
    }
}
```

Output:

```
Smith
Cooper
Baker
Fletcher
Miller
```

Disarium numbers

```
func is_disarium(n) {
    n.digits.flip.sum_kv{|k,d| d**(k+1) } == n
}

say 18.by(is_disarium)
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 89, 135, 175, 518, 598, 1306, 1676, 2427]
```

Discordian date

```
require('Time::Piece');

var seasons = %w(Chaos Discord Confusion Bureaucracy The\ Aftermath);
var week_days = %w(Sweetmorn Boomtime Pungenday Prickle-Prickle Setting\ Orange);

func ordinal (n) {
    "#{n}" + (n % 100 ~~ [11,12,13] ? 'th'
                                    : <th st nd rd th th th th th th>[n % 10])
}

func ddate(ymd) {
    var d = %s'Time::Piece'.strptime(ymd, '%Y-%m-%d');
    var yold = "in the YOLD #{d.year + 1166}";

    var day_of_year0 = d.day_of_year;

    if (d.is_leap_year) {
        return "St. Tib's Day, #{yold}" if ([d.mon, d.mday] == [2, 29]);
        day_of_year0-- if (day_of_year0 >= 60); # Compensate for St. Tib's Day
    }

    var weekday = week_days[day_of_year0 % week_days.len];
    var season = seasons[day_of_year0 / 73];
    var season_day = ordinal(day_of_year0 % 73 + 1);

    return "#{weekday}, the #{season_day} day of #{season} #{yold}";
}

%w(2010-07-22 2012-02-28 2012-02-29 2012-03-01).each { |ymd|
    say "#{ymd} is #{ddate(ymd)}"
}
```

Output:

```
2010-07-22 is Pungenday, the 57th day of Confusion in the YOLD 3176
2012-02-28 is Prickle-Prickle, the 59th day of Chaos in the YOLD 3178
2012-02-29 is St. Tib's Day, in the YOLD 3178
2012-03-01 is Setting Orange, the 60th day of Chaos in the YOLD 3178
```

Display a linear combination

```
func linear_combination(coeffs) {
    var res = ""
    for e,f in (coeffs.kv) {
        given(f) {
            when (1) {
                res += "+e(#{e+1})"
            }
            when (-1) {
                res += "-e(#{e+1})"
            }
            case (.> 0) {
                res += "+#{f}*e(#{e+1})"
            }
            case (.< 0) {
                res += "#{f}*e(#{e+1})"
            }
        }
    }
    res -= /^\+/
    res || 0
}

var tests = [
    %n{1 2 3},
    %n{0 1 2 3},
    %n{1 0 3 4},
    %n{1 2 0},
    %n{0 0 0},
    %n{0},
    %n{1 1 1},
    %n{-1 -1 -1},
    %n{-1 -2 0 -3},
    %n{-1},
]

tests.each { |t|
    printf("%10s -> %-10s\n", t.join(' '), linear_combination(t))
}
```

Output:

```
     1 2 3 -> e(1)+2*e(2)+3*e(3)
   0 1 2 3 -> e(2)+2*e(3)+3*e(4)
   1 0 3 4 -> e(1)+3*e(3)+4*e(4)
     1 2 0 -> e(1)+2*e(2)
     0 0 0 -> 0
         0 -> 0
     1 1 1 -> e(1)+e(2)+e(3)
  -1 -1 -1 -> -e(1)-e(2)-e(3)
-1 -2 0 -3 -> -e(1)-2*e(2)-3*e(4)
        -1 -> -e(1)
```

Distinct palindromes within decimal numbers

```
func palindromes(arr) {
    gather {
        for a in (0..arr.end), b in (a .. arr.end) {
            var sublist = arr.ft(a, b)
            take(sublist) if (sublist == sublist.flip)
        }
    }.uniq
}

for n in (100..125) {
    say "#{n} -> #{palindromes(n.digits).sort.map{.join}.sort_by{.len}.join(' ')}"
}

[9, 169, 12769, 1238769, 123498769, 12346098769, 1234572098769,
 123456832098769, 12345679432098769, 1234567905432098769, 12345679016543209876 9,
 83071934127905179083, 13202679478494903612056 95, "amanaplanacanalpanama"].each {|n|
    var p = palindromes(n.kind_of(Number) ? n.digits : n.chars).grep { .len >= 2}
    say ("#{'%25s' % n} has #{'%2d' % p.len} palindromes of length 2 or more: ",
        p.sort.map{.join}.sort_by{.len}.join(' '))
}
```

Output:

```
100 -> 0 1 00
101 -> 0 1 101
102 -> 0 1 2
103 -> 0 1 3
104 -> 0 1 4
105 -> 0 1 5
106 -> 0 1 6
107 -> 0 1 7
108 -> 0 1 8
109 -> 0 1 9
110 -> 0 1 11
111 -> 1 11 111
112 -> 1 2 11
113 -> 1 3 11
114 -> 1 4 11
115 -> 1 5 11
116 -> 1 6 11
117 -> 1 7 11
118 -> 1 8 11
119 -> 1 9 11
120 -> 0 1 2
121 -> 1 2 121
122 -> 1 2 22
123 -> 1 2 3
124 -> 1 2 4
125 -> 1 2 5
                        9 has  0 palindromes of length 2 or more:
                      169 has  0 palindromes of length 2 or more:
                    12769 has  0 palindromes of length 2 or more:
                  1238769 has  0 palindromes of length 2 or more:
                123498769 has  0 palindromes of length 2 or more:
              12346098769 has  0 palindromes of length 2 or more:
            1234572098769 has  0 palindromes of length 2 or more:
          123456832098769 has  0 palindromes of length 2 or more:
        12345679432098769 has  0 palindromes of length 2 or more:
      1234567905432098769 has  0 palindromes of length 2 or more:
    12345790165432098769 has  0 palindromes of length 2 or more:
     83071934127905179083 has  0 palindromes of length 2 or more:
1320267947849490361205695 has  3 palindromes of length 2 or more: 202 494 949
    amanaplanacanalpanama has 12 palindromes of length 2 or more: aca ama ana nacan anacana lanacanal p
```

Distinct power numbers

```
[2..5]*2 -> cartesian.map_2d {|a,b| a**b }.sort.uniq.say
```

Alternative solution:

```
2..5 ~X** 2..5 -> sort.uniq.say
```

Output:

```
[4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125]
```

Distribution of 0 digits in factorial series

```
func mean_factorial_digits(n, d = 0) {

    var v = 1
    var total = 0.float

    for k in (1..n) {
        v *= k
        total += v.digits.count(d)/v.len
    }

    total / n
}

say mean_factorial_digits(100)
say mean_factorial_digits(1000)
say mean_factorial_digits(10000)
```

Output:

```
0.24675318616743221777841588719735269911294070332 7
0.20354455110316463564004380317114553029857411678 9
0.17300384824186605318003664289307061568102788090 6
```

Diversity prediction theorem

```
func avg_error(m, v) {
    v.map { (_ - m)**2 }.sum / v.len
}

func diversity_calc(truth, pred) {
    var ae = avg_error(truth, pred)
    var cp = pred.sum/pred.len
    var ce = (cp - truth)**2
    var pd = avg_error(cp, pred)
    return [ae, ce, pd]
}

func diversity_format(stats) {
    gather {
        for t,v in (%w(average-error crowd-error diversity) ~Z stats) {
            take(("%13s" % t) + ':' + ('%7.3f' % v))
        }
    }
}

diversity_format(diversity_calc(49, [48, 47, 51])).each{.say}
diversity_format(diversity_calc(49, [48, 47, 51, 42])).each{.say}
```

Output:

```
average-error:  3.000
  crowd-error:  0.111
    diversity:  2.889
average-error: 14.500
  crowd-error:  4.000
    diversity: 10.500
```

DNS query

```
var (err, *res) = Socket.getaddrinfo(
        'www.kame.net', 0,
        Hash(protocol => Socket.IPPROTO_TCP)
)
err && die err
for z (res) {
    say [Socket.getnameinfo(z{:addr}, Socket.NI_NUMERICHOST)][1]
}
```

Output:

```
203.178.141.194
2001:200:dff:fff1:216:3eff:feb1:44d7
```

Dot product

```
func dot_product(a, b) {
    (a »*« b)«+»;
}
say dot_product([1,3,-5], [4,-2,-1]);   # => 3
```

Doubly-linked list/Element definition

```
var node = Hash(
     data => 'say what',
     next => foo_node,
     prev => bar_node,
)

node{:next} = quux_node  # mutable
```

Dragon curve

Uses the LSystem class defined at Hilbert curve.

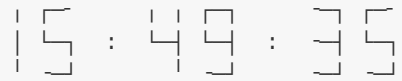
```
var rules = Hash(
    x => 'x+yF+',
    y => '-Fx-y',
)

var lsys = LSystem(
    width:  600,
    height: 600,

    xoff: -430,
    yoff: -380,

    len:   8,
    angle: 90,
    color: 'dark green',
)

lsys.execute('Fx', 11, "dragon_curve.png", rules)
```

Output image: Dragon curve

Draw a clock

```
STDOUT.autoflush(true)

var (rows, cols) = `stty size`.nums...

var x = (rows/2 - 1  -> int)
var y = (cols/2 - 16 -> int)

var chars = [
                "┌┐  ╷ ╶┐ ╶┐ ╷┌╴╶┐ ╶┐┌┐┌┐   ",
                "│ │  │┌╴┤ └┤ └┤└┐  ├┤└┤ : ",
                "└┘  ╵└─ └┘  ╵ └┘└┘  ╵└┘└┘  "
            ].map {|s| s.split(3) }

func position(i,j) {
    "\e[%d;%dH" % (i, j)
}

func indices {
    var t = Time.local
    "%02d:%02d:%02d" % (t.hour, t.min, t.sec) -> split(1).map{|c| c.ord - '0'.ord }
}

loop {
    print "\e[H\e[J"
    for i in ^chars {
      print position(x + i, y)
      print [chars[i][indices()]].join(' ')
    }
    print position(1, 1)
    Sys.sleep(0.1)
}
```

Output:



Draw a cuboid

```
const dirs = Hash("-" => [1,0], "|" => [0,1], "/" => [1,1])

func cuboid(nx, ny, nz) {
  say("cuboid %d %d %d:" % [nx, ny, nz])
  var(x, y, z) = (8*nx, 2*ny, 4*nz)
  var area = []
  var line = func(n, sx, sy, c) {
    var(dx, dy) = dirs{c}...;
    for i (0..n) {
      var (xi, yi) = (sx + i*dx, sy + i*dy)
      area[yi] \\= [" "]*(x+y+1)
      area[yi][xi] = (area[yi][xi] == " " ? c : '+')
    }
  }

  0 .. nz-1 -> each {|i| line.call(x,       0,     4*i, "-")}
  0 .. ny   -> each {|i| line.call(x,     2*i, z + 2*i, "-")}
  0 .. nx-1 -> each {|i| line.call(z,     8*i,       0, "|")}
  0 .. ny   -> each {|i| line.call(z, x + 2*i,     2*i, "|")}
  0 .. nz-1 -> each {|i| line.call(y,       x,     4*i, "/")}
  0 .. nx   -> each {|i| line.call(y,     8*i,       z, "/")}

  area.reverse.each { |line|
     say line.join('')
  }
}

cuboid(2, 3, 4)
cuboid(1, 1, 1)
cuboid(6, 2, 1)
cuboid(2, 4, 1)
```

A faster approach:

```
func cuboid (x=1,y=1,z=1,s=' ',c='+',h='-',v='|',d='/') {
    say("cuboid %d %d %d:" % (x, y, z))
    ' ' * z+1 + c + h*x + c -> say

    { |i|
        ' ' * (z - i + 1) + d + s*x + d +
              (s * (i - (i > y ? i-y : 1))) +
              (i - 1 == y ? c : (i > y ? d : v)) -> say
    }.for(1..z)

    c + h*x + c + (s * (z < y ? z : y) +
        (z < y ? v : (z == y ? c : d))) -> say

    { |i|
        v + s*x + v + (z > y
            ? (i >= z ? (s*x + c) : (s * y-i + d))
            : (y - i > z
                ? (s * z + v)
                : (s * y-i + (y-i == z ? c : d))
              )
        ) -> say;
    }.for(1..y)

    c + h*x + c -> say
}

cuboid(2, 3, 4)
cuboid(1, 1, 1)
cuboid(6, 2, 1)
cuboid(2, 4, 1)
```

Output:

```
cuboid 2 3 4:
     +--+
    /  /|
   /  / |
  /  /  |
 /  /   +
+--+   /
|  |  /
|  | /
|  |/
+--+
cuboid 1 1 1:
  +-+
 / /|
+-+ +
| |/
+-+
cuboid 6 2 1:
  +------+
 /      /|
+------+ |
|      | +
|      |/
+------+
cuboid 2 4 1:
  +--+
 /  /|
+--+ |
|  | |
|  | |
|  | +
|  |/
+--+
```

Draw a sphere

Produces a PGM image.

```
func normalize (vec)  { vec »/» (vec »*« vec -> sum.sqrt) }
func dot       (x, y) { -(x »*« y -> sum) `max` 0 }

var x = var y = 255
x += 1 if x.is_even    # must be odd

var light = normalize([ 3, 2, -5 ])
var depth = 255

func draw_sphere(rad, k, ambient) {
    var pixels = []
    var r2 = (rad * rad)
    var range = (-rad .. rad)
    for x,y in (range ~X range) {
        if ((var x2 = x*x) + (var y2 = y*y) < r2) {
            var vector = normalize([x, y, (r2 - x2 - y2).sqrt])
            var intensity = (dot(light, vector)**k + ambient)
            var pixel = (0 `max` (intensity*depth -> int) `min` depth)
            pixels << pixel
        }
        else {
            pixels << 0
        }
    }
    return pixels
}

var outfile = %f'sphere-sidef.pgm'
var out = outfile.open('>:raw')

out.say("P5\n#{x} #{y}\n#{depth}")    # .pgm header
out.print(draw_sphere((x-1)/2, .9, .2).map{.chr}.join)
out.close
```

Duffinian numbers

```
func is_duffinian(n) {
    n.is_composite && n.is_coprime(n.sigma)
}

say "First 50 Duffinian numbers:"
say 50.by(is_duffinian)

say "\nFirst 15 Duffinian triplets:"
15.by{|n| ^3 -> all {|k| is_duffinian(n+k) } }.each {|n|
    printf("(%s, %s, %s)\n", n, n+1, n+2)
}
```

Output:

```
First 50 Duffinian numbers:
[4, 8, 9, 16, 21, 25, 27, 32, 35, 36, 39, 49, 50, 55, 57, 63, 64, 65, 75, 77, 81, 85, 93, 98, 100, 111,

First 15 Duffinian triplets:
(63, 64, 65)
(323, 324, 325)
(511, 512, 513)
(721, 722, 723)
(899, 900, 901)
(1443, 1444, 1445)
(2303, 2304, 2305)
(2449, 2450, 2451)
(3599, 3600, 3601)
(3871, 3872, 3873)
(5183, 5184, 5185)
(5617, 5618, 5619)
(6049, 6050, 6051)
(6399, 6400, 6401)
(8449, 8450, 8451)
```

Dynamic variable names

It is not possible to create a new lexical variable at run-time, but there are other various ways to do something similar.

```
var name = read("Enter a variable name: ", String);     # type in 'foo'

class DynamicVar(name, value) {
    method init {
        DynamicVar.def_method(name, ->(_) { value })
    }
}

var v = DynamicVar(name, 42);       # creates a dynamic variable
say v.foo;                          # retrieves the value
```

Earliest difference between prime gaps

```
func prime_gap_records(upto) {

    var gaps = []
    var p = 3

    each_prime(p.next_prime, upto, {|q|
        gaps[q-p] := p
        p = q
    })

    gaps.grep { defined(_) }
}

var gaps = prime_gap_records(1e8)

for m in (1 .. gaps.max.len) {
    gaps.each_cons(2, {|p,q|
        if (abs(q-p) > 10**m) {
            say "10^#{m} -> (#{p}, #{q}) with gaps (#{
            p.next_prime-p}, #{q.next_prime-q}) and difference #{abs(q-p)}"
            break
        }
    })
}
```

Output:

```
10^1 -> (7, 23) with gaps (4, 6) and difference 16
10^2 -> (113, 1831) with gaps (14, 16) and difference 1718
10^3 -> (113, 1831) with gaps (14, 16) and difference 1718
10^4 -> (9551, 30593) with gaps (36, 38) and difference 21042
10^5 -> (173359, 31397) with gaps (70, 72) and difference 141962
10^6 -> (396733, 1444309) with gaps (100, 102) and difference 1047576
10^7 -> (2010733, 13626257) with gaps (148, 150) and difference 11615524
```

Egyptian division

```
func egyptian_divmod(dividend, divisor) {
  var table = [[1, divisor]]
  table << table[-1].map{|e| 2*e } while (2*table[-1][0] <= dividend)
  var (answer, accumulator) = (0, 0)
  table.reverse.each { |pair|
    var (pow, double) = pair...
    if (accumulator + double <= dividend) {
      accumulator += double
      answer += pow
    }
  }
  return (answer, dividend - accumulator)
}

say ("Quotient = %s Remainder = %s" % egyptian_divmod(580, 34))
```

Output:

```
Quotient = 17 Remainder = 2
```

Egyptian fractions

```
func ef(fr) {
  var ans = []
  if (fr >= 1) {
    return([fr]) if (fr.is_int)
    var intfr = fr.to_i
    ans << intfr
    fr -= intfr
  }
  var (x, y) = fr.nude
  while (x != 1) {
    ans << fr.inv.ceil.inv
    fr = ((-y % x) / y*fr.inv.ceil)
    (x, y) = fr.nude
  }
  ans << fr
  return ans
}

for fr in [43/48, 5/121, 2014/59] {
  "%s => %s\n".printf(fr.as_rat, ef(fr).map{.as_rat}.join(' + '))
}

var lenmax = (var denommax = [0])
for b in (2 .. 99) {
  for a in (1 .. b-1) {
    var fr = a/b
    var e = ef(fr)
    var (elen, edenom) = (e.length, e[-1].denominator)
    lenmax = [elen, fr] if (elen > lenmax[0])
    denommax = [edenom, fr] if (edenom > denommax[0])
  }
}

"Term max is %s with %i terms\n".printf(lenmax[1].as_rat, lenmax[0])
"Denominator max is %s with %i digits\n".printf(denommax[1].as_rat, denommax[0].size)
say denommax[0]
```

Output:

```
43/48 => 1/2 + 1/3 + 1/16
5/121 => 1/25 + 1/757 + 1/763309 + 1/873960180913 + 1/1527612795642093418846225
2014/59 => 34 + 1/8 + 1/95 + 1/14947 + 1/670223480
Term max is 44/53 with 8 terms
Denominator max is 8/97 with 150 digits
579504587067542801713103191859918608251030291952195423583529357653899418686342360361798689053273749372 6
```

EKG sequence convergence

```
class Seq(terms, callback) {
    method next {
        terms += callback(terms)
    }

    method nth(n) {
        while (terms.len < n) {
            self.next
        }
        terms[n-1]
    }

    method first(n) {
        while (terms.len < n) {
            self.next
        }
        terms.first(n)
    }
}

func next_EKG (s) {
    2..Inf -> first {|k|
        !(s.contains(k) || s[-1].is_coprime(k))
    }
}

func EKG (start) {
    Seq([1, start], next_EKG)
}

func converge_at(ints) {
    var ekgs = ints.map(EKG)

    2..Inf -> first {|k|
        (ekgs.map { .nth(k)        }.uniq.len == 1) &&
        (ekgs.map { .first(k).sort }.uniq.len == 1)
    }
}

for k in [2, 5, 7, 9, 10] {
    say "EKG(#{k}) = #{EKG(k).first(10)}"
}

for arr in [[5,7], [2, 5, 7, 9, 10]] {
    var c = converge_at(arr)
    say "EKGs of #{arr} converge at term #{c}"
}
```

Output:

```
EKG(2) = [1, 2, 4, 6, 3, 9, 12, 8, 10, 5]
EKG(5) = [1, 5, 10, 2, 4, 6, 3, 9, 12, 8]
EKG(7) = [1, 7, 14, 2, 4, 6, 3, 9, 12, 8]
EKG(9) = [1, 9, 3, 6, 2, 4, 8, 10, 5, 15]
EKG(10) = [1, 10, 2, 4, 6, 3, 9, 12, 8, 14]
EKGs of [5, 7] converge at term 21
EKGs of [2, 5, 7, 9, 10] converge at term 45
```

Element-wise operations

The built-in metaoperators `~W<op>`, `~S<op>` and `~RS<op>` are defined for arbitrary nested arrays.

```
var m1 = [[3,1,4],[1,5,9]]
var m2 = [[2,7,1],[8,2,2]]

say ":: Matrix-matrix operations"
say (m1 ~W+  m2)
say (m1 ~W-  m2)
say (m1 ~W*  m2)
say (m1 ~W/  m2)
say (m1 ~W// m2)
say (m1 ~W** m2)
say (m1 ~W%  m2)

say "\n:: Matrix-scalar operations"
say (m1 ~S+  42)
say (m1 ~S-  42)
say (m1 ~S/  42)
say (m1 ~S** 10)
# ...

say "\n:: Scalar-matrix operations"
say (m1 ~RS+  42)
say (m1 ~RS-  42)
say (m1 ~RS/  42)
say (m1 ~RS** 10)
# ...
```

Output:

```
:: Matrix-matrix operations
[[5, 8, 5], [9, 7, 11]]
[[1, -6, 3], [-7, 3, 7]]
[[6, 7, 4], [8, 10, 18]]
[[3/2, 1/7, 4], [1/8, 5/2, 9/2]]
[[1, 0, 4], [0, 2, 4]]
[[9, 1, 4], [1, 25, 81]]
[[1, 1, 0], [1, 1, 1]]

:: Matrix-scalar operations
[[45, 43, 46], [43, 47, 51]]
[[-39, -41, -38], [-41, -37, -33]]
[[1/14, 1/42, 2/21], [1/42, 5/42, 3/14]]
[[59049, 1, 1048576], [1, 9765625, 3486784401]]

:: Scalar-matrix operations
[[45, 43, 46], [43, 47, 51]]
[[39, 41, 38], [41, 37, 33]]
[[14, 42, 21/2], [42, 42/5, 14/3]]
[[1000, 10, 10000], [10, 100000, 1000000000]]
```

Elementary cellular automaton

```
class Automaton(rule, cells) {

    method init {
        rule = sprintf("%08b", rule).chars.map{.to_i}.reverse
    }

    method next {
        var previous = cells.map{_}
        var len = previous.len
        cells[] = rule[
                    previous.range.map { |i|
                        4*previous[i-1 % len] +
                        2*previous[i]         +
                        previous[i+1 % len]
                    }
                  ]
    }

    method to_s {
        cells.map { _ ? '#' : ' ' }.join
    }
}

var size = 20
var arr = size.of(0)
arr[size/2] = 1

var auto = Automaton(90, arr)

(size/2).times {
    print "|#{auto}|\n"
    auto.next
}
```

Output:

```
|          #          |
|         # #         |
|        #   #        |
|       # # # #       |
|      #       #      |
|     # #     # #     |
|    #   #   #   #    |
|   # # # # # # # #   |
|  #               # |
| # #             # #|
```

Elementary cellular automaton/Infinite length

```
func evolve(rule, bin) {
    var offset = 0
    var (l='', r='')
    Inf.times {
        bin.sub!(/^((.)\g2*)/, {|_s1, s2| l = s2; offset -= s2.len; s2*2 })
        bin.sub!(/(.)\g1*$/, {|s1| r = s1; s1*2 })
        printf("%5d| %s%s\n", offset, ' ' * (40 + offset), bin.tr('01','.#'))
        bin = [l*3, 0.to(bin.len-3).map{|i| bin.substr(i, 3) }..., r*3 ].map { |t|
                1 & (rule >> t.bin)
        }.join
    }
}

evolve(90, "010")
```

Output:

```
   -1|                                        ..#..
   -2|                                       ..#.#..
   -3|                                      ..#...#..
   -4|                                     ..#.#.#.#..
   -5|                                    ..#.......#..
   -6|                                   ..#.#.....#.#..
   -7|                                  ..#...#...#...#..
   -8|                                 ..#.#.#.#.#.#.#.#..
   -9|                                ..#..............#..
  -10|                               ..#.#............#.#..
  -11|                              ..#...#..........#...#..
  -12|                             ..#.#.#.#........#.#.#.#..
  -13|                            ..#.......#.......#.......#..
  -14|                           ..#.#.....#.#.....#.#.....#.#..
  -15|                          ..#...#...#...#...#...#...#...#..
  -16|                         ..#.#.#.#.#.#.#.#.#.#.#.#.#.#.#..
  -17|                        ..#.............................#..
  -18|                       ..#.#...........................#.#..
  -19|                      ..#...#.........................#...#..
  -20|                     ..#.#.#.#.......................#.#.#.#..
   ⋮
```

Elementary cellular automaton/Random number generator

```
var auto = Automaton(30, [1] + 100.of(0));

10.times {
    var sum = 0;
    8.times {
        sum = (2*sum + auto.cells[0]);
        auto.next;
    };
    say sum;
};
```

Output:

```
220
197
147
174
117
97
149
171
240
241
```

Elliptic curve arithmetic

```
module EC {

    var A = 0
    var B = 7

    class Horizon {
        method to_s {
            "EC Point at horizon"
        }

        method *(_) {
            self
        }

        method -(_) {
            self
        }
    }

    class Point(Number x, Number y) {
        method to_s {
            "EC Point at x=#{x}, y=#{y}"
        }

        method neg {
            Point(x, -y)
        }

        method -(Point p) {
            self + -p
        }

        method +(Point p) {

            if (x == p.x) {
                return (y == p.y ? self*2 : Horizon())
            }
            else {
                var slope = (p.y - y)/(p.x - x)
                var x2 = (slope**2 - x - p.x)
                var y2 = (slope * (x - x2) - y)
                Point(x2, y2)
            }
        }

        method +(Horizon _) {
            self
        }
```

```
        method *((0)) {
            Horizon()
        }

        method *((1)) {
            self
        }

        method *((2)) {
            var l = (3 * x**2 + A)/(2 * y)
            var x2 = (l**2 - 2*x)
            var y2 = (l * (x - x2) - y)
            Point(x2, y2)
        }

        method *(Number n) {
            2*(self * (n>>1)) + self*(n % 2)
        }
    }

    class Horizon {
        method +(Point p) {
            p
        }
    }

    class Number {
        method +(Point p) {
            p + self
        }
        method *(Point p) {
            p * self
        }
        method *(Horizon h) {
            h
        }
        method -(Point p) {
            -p + self
        }
    }
}

say var p = with(1) {|v| EC::Point(v, sqrt(abs(1 - v**3 - EC::A*v - EC::B))) }
say var q = with(2) {|v| EC::Point(v, sqrt(abs(1 - v**3 - EC::A*v - EC::B))) }
say var s = (p + q)

say ("checking alignment:  ", abs((p.x - q.x)*(-s.y - q.y) - (p.y - q.y)*(s.x - q.x)) < 1e-20)
```

Output:

```
EC Point at x=1, y=2.64575131106459059050161575363926042571025918308
EC Point at x=2, y=3.74165738677394138558374873231654930175601980778
EC Point at x=-1.79898987322333068322364213893577309997540625528, y=0.42167869684980302897488245831443O
checking alignment:  true
```

Emirp primes

```
func forprimes(a, b, callback) {
    for (var p = a.dec.next_prime; p <= b; p.next_prime!) {
        callback(p)
    }
}

func is_emirp(p) {
    var str = Str(p)
    var rev = str.reverse
    (str != rev) && is_prime(Num(rev))
}

func emirp_list(count) {
    var i = 13
    var inc = (100 + 10*count)
    var n = []
    while (n.len < count) {
        forprimes(i, i+inc - 1, {|p|
            is_emirp(p) && (n << p)
        })
        (i, inc) = (i+inc, int(inc * 1.03) + 1000)
    }
    n.splice(count)
    return n
}

say ("First 20: ", emirp_list(20).join(' '))
say ("Between 7700 and 8000: ", gather {
        forprimes(7700, 8000, {|p| is_emirp(p) && take(p) })
    }.join(' '))
say ("The 10,000'th emirp: ", emirp_list(10000)[-1])
```

Output:

```
First 20: 13 17 31 37 71 73 79 97 107 113 149 157 167 179 199 311 337 347 359 389
Between 7700 and 8000: 7717 7757 7817 7841 7867 7879 7901 7927 7949 7951 7963
The 10,000'th emirp: 948349
```

Empty directory

Built-in method:

```
Dir('/my/dir').is_empty;    # true, false or nil
```

User-defined function:

```
func is_empty(dir) {
    dir.open(\var dir_h) || return nil;
    dir_h.each { |file|
        file ~~ ['.', '..'] && next;
        return false;
    };
    return true;
};
```

Empty program

```
```

Empty string

Create an empty string:

```
var s = "";
var s = String.new;
```

These expressions all evaluate to true to determine emptiness:

```
s == "";
s.length == 0;
s.is_empty;
s ~~ /^\z/;
s ~~ /\A\z/;
```

Non-empty expressions, in addition to simply negating the above expressions:

```
s != "";
s.length > 0;
s ~~ /./s;
s !~ /^\z/;
```

Enforced immutability

```
define PI = 3.14159;            # compile-time defined constant
const MSG = "Hello world!";     # run-time defined constant
```

Entropy

```
func entropy(s) {
  var counts = Hash.new;
  s.each { |c| counts{c} := 0 ++ };
  var len = s.len;
  [0, counts.values.map {|count|
    var freq = count/len; freq * freq.log2 }...
  ]«-»;
}

say entropy("1223334444");
```

Output:

```
1.846439344671015493434197746305045223237
```

Entropy/Narcissist

```
func entropy(s) {
    [0,
        s.chars.freq.values.map {|c|
            var f = c/s.len
            f * f.log2
        }...
    ]«-»
}

say entropy(File(__FILE__).open_r.slurp)
```

Output:

```
4.273077508664349157134321091865 49
```

Enumerations

Implicit:

```
enum {Apple, Banana, Cherry};   # numbered 0 through 2
```

Explicit:

```
enum {
    Apple=3,
    Banana,         # gets the value 4
    Cherry="a",
    Orange,         # gets the value "b"
};
```

Environment variables

The *ENV* hash maps environment variables to their values:

```
say ENV{'HOME'};
```

Equal prime and composite sums

```
func f(n) {

    var (
        p = 2, sp = p,
        c = 4, sc = c,
    )

    var res = []

    while (res.len < n) {
        if (sc == sp) {
            res << [sp, c.composite_count, p.prime_count]
            sc += c.next_composite!
        }
        while (sp < sc) {
            sp += p.next_prime!
        }
        while (sc < sp) {
            sc += c.next_composite!
        }
    }

    return res
}

f(8).each_2d {|n, ci, pi|
    printf("%12s = %-9s = %s\n", n, "P(#{pi})", "C(#{ci})")
}
```

Output:

```
          10 = P(3)      = C(2)
        1988 = P(33)     = C(51)
       14697 = P(80)     = C(147)
       83292 = P(175)    = C(361)
     1503397 = P(660)    = C(1582)
    18859052 = P(2143)   = C(5699)
    93952013 = P(4556)   = C(12821)
 89171409882 = P(118785) = C(403341)
```

(takes ~6 seconds)

Equilibrium index

```
func eq_index(nums) {
    var (i, sum, sums) = (0, 0, Hash.new);
    nums.each { |n|
        sums{2*sum + n} := [] -> append(i++);
        sum += n;
    };
    sums{sum} \\ [];
}
```

Test:

```
var indices = [
  [-7, 1, 5, 2,-4, 3, 0],
  [2, 4, 6],
  [2, 9, 2],
  [1,-1, 1,-1, 1,-1, 1],
]

indices.each { |x|
    say ("%s => %s" % @|[x, eq_index(x)].map{.dump});
}
```

Output:

```
[-7, 1, 5, 2, -4, 3, 0] => [3, 6]
[2, 4, 6] => []
[2, 9, 2] => [1]
[1, -1, 1, -1, 1, -1, 1] => [0, 1, 2, 3, 4, 5, 6]
```

Erdös-Selfridge categorization of primes

```
func Erdös_Selfridge_class(n, s=1) is cached {
    var f = factor_exp(n+s)
    f.last.head > 3 || return 1
    f.map {|p| __FUNC__(p.head, s) }.max + 1
}

say "First two hundred primes; Erdös-Selfridge categorized:"
200.pn_primes.group_by(Erdös_Selfridge_class).sort_by{.to_i}.each_2d {|k,v|
    say "#{k} => #{v}"
}

say "\nSummary of first 10^6 primes; Erdös-Selfridge categorized:";
1e6.pn_primes.group_by(Erdös_Selfridge_class).sort_by{.to_i}.each_2d {|k,v|
    printf("Category %2d: first: %9s  last: %10s  count: %s\n", k, v.first, v.last, v.len)
}
```

Output:

```
First two hundred primes; Erdös-Selfridge categorized:
1 => [2, 3, 5, 7, 11, 17, 23, 31, 47, 53, 71, 107, 127, 191, 383, 431, 647, 863, 971, 1151]
2 => [13, 19, 29, 41, 43, 59, 61, 67, 79, 83, 89, 97, 101, 109, 131, 137, 139, 149, 167, 179, 197, 199,
3 => [37, 103, 113, 151, 157, 163, 173, 181, 193, 227, 233, 257, 277, 311, 331, 337, 347, 353, 379, 389
4 => [73, 313, 443, 617, 661, 673, 677, 691, 739, 757, 823, 887, 907, 941, 977, 1093, 1109, 1129, 1201,
5 => [1021]

Summary of first 10^6 primes; Erdös-Selfridge categorized:
Category  1: first:         2  last:   10616831  count: 46
Category  2: first:        13  last:   15482669  count: 10497
Category  3: first:        37  last:   15485863  count: 201987
Category  4: first:        73  last:   15485849  count: 413891
Category  5: first:      1021  last:   15485837  count: 263109
Category  6: first:      2917  last:   15485857  count: 87560
Category  7: first:     15013  last:   15484631  count: 19389
Category  8: first:     49681  last:   15485621  count: 3129
Category  9: first:    532801  last:   15472811  count: 363
Category 10: first:   1065601  last:   15472321  count: 28
Category 11: first:   8524807  last:    8524807  count: 1
```

Erdős-primes

```
func is_erdos_prime(p) {

    return true  if p==2
    return false if !p.is_prime

    var f = 1

    for (var k = 2; f < p; k++) {
        p - f -> is_composite || return false
        f *= k
    }

    return true
}

say ("Erdős primes <= 2500: ", 1..2500 -> grep(is_erdos_prime))
say ("The 7875th Erdős prime is: ", is_erdos_prime.nth(7875))
```

Output:

```
Erdős primes <= 2500: [2, 101, 211, 367, 409, 419, 461, 557, 673, 709, 769, 937, 967, 1009, 1201, 1259,
The 7875th Erdős prime is: 999721
```

Esthetic numbers

```
func generate_esthetic(root, upto, callback, b=10) {

    var v = root.digits2num(b)

    return nil if (v > upto)
    callback(v)

    var t = root.head

    __FUNC__([t+1, root...], upto, callback, b) if (t+1  < b)
    __FUNC__([t-1, root...], upto, callback, b) if (t-1 >= 0)
}

func between_esthetic(from, upto, b=10) {
    gather {
        for k in (1..^b) {
            generate_esthetic([k], upto, { take(_) if (_ >= from) }, b)
        }
    }.sort
}

func first_n_esthetic(n, b=10) {
    for (var m = n**2; true ; m *= b) {
        var list = between_esthetic(1, m, b)
        return list.first(n) if (list.len >= n)
    }
}

for b in (2..16) {
    say "\n#{b}-esthetic numbers with indices #{4*b}..#{6*b}: "
    say first_n_esthetic(6*b, b).last(6*b - 4*b + 1).map{.base(b)}.join(' ')
}

say "\nBase 10 esthetic numbers between 1,000 and 9,999:"
between_esthetic(1e3, 1e4).slices(20).each { .join(' ').say }

say "\nBase 10 esthetic numbers between 100,000,000 and 130,000,000:"
between_esthetic(1e8, 13e7).slices(9).each { .join(' ').say }
```

Output:

```
2-esthetic numbers with indices 8..12:
10101010 101010101 1010101010 10101010101 101010101010

3-esthetic numbers with indices 12..18:
1210 1212 2101 2121 10101 10121 12101

4-esthetic numbers with indices 16..24:
323 1010 1012 1210 1212 1232 2101 2121 2123

5-esthetic numbers with indices 20..30:
323 343 432 434 1010 1012 1210 1212 1232 1234 2101

6-esthetic numbers with indices 24..36:
343 345 432 434 454 543 545 1010 1012 1210 1212 1232 1234

7-esthetic numbers with indices 28..42:
345 432 434 454 456 543 545 565 654 656 1010 1012 1210 1212 1232

8-esthetic numbers with indices 32..48:
432 434 454 456 543 545 565 567 654 656 676 765 767 1010 1012 1210 1212

9-esthetic numbers with indices 36..54:
```

```
434 454 456 543 545 565 567 654 656 676 678 765 767 787 876 878 1010 1012 1210

10-esthetic numbers with indices 40..60:
454 456 543 545 565 567 654 656 676 678 765 767 787 789 876 878 898 987 989 1010 1012

11-esthetic numbers with indices 44..66:
456 543 545 565 567 654 656 676 678 765 767 787 789 876 878 898 89a 987 989 9a9 a98 a9a 1010

12-esthetic numbers with indices 48..72:
543 545 565 567 654 656 676 678 765 767 787 789 876 878 898 89a 987 989 9a9 9ab a98 a9a aba ba9 bab

13-esthetic numbers with indices 52..78:
545 565 567 654 656 676 678 765 767 787 789 876 878 898 89a 987 989 9a9 9ab a98 a9a aba abc ba9 bab bcb

14-esthetic numbers with indices 56..84:
565 567 654 656 676 678 765 767 787 789 876 878 898 89a 987 989 9a9 9ab a98 a9a aba abc ba9 bab bcb bcd

15-esthetic numbers with indices 60..90:
567 654 656 676 678 765 767 787 789 876 878 898 89a 987 989 9a9 9ab a98 a9a aba abc ba9 bab bcb bcd cba

16-esthetic numbers with indices 64..96:
654 656 676 678 765 767 787 789 876 878 898 89a 987 989 9a9 9ab a98 a9a aba abc ba9 bab bcb bcd cba cbc

Base 10 esthetic numbers between 1,000 and 9,999:
1010 1012 1210 1212 1232 1234 2101 2121 2123 2321 2323 2343 2345 3210 3212 3232 3234 3432 3434 3454
3456 4321 4323 4343 4345 4543 4545 4565 4567 5432 5434 5454 5456 5654 5656 5676 5678 6543 6545 6565
6567 6765 6767 6787 6789 7654 7656 7676 7678 7876 7878 7898 8765 8767 8787 8789 8987 8989 9876 9878
9898

Base 10 esthetic numbers between 100,000,000 and 130,000,000:
101010101 101010121 101010123 101012101 101012121 101012123 101012321 101012323 101012343
101012345 101210101 101210121 101210123 101212101 101212121 101212123 101212321 101212323
101212343 101212345 101232101 101232121 101232123 101232321 101232323 101232343 101232345
101234321 101234323 101234343 101234345 101234543 101234545 101234565 101234567 121010101
121010121 121010123 121012101 121012121 121012123 121012321 121012323 121012343 121012345
121210101 121210121 121210123 121212101 121212121 121212123 121212321 121212323 121212343
121212345 121232101 121232121 121232123 121232321 121232323 121232343 121232345 121234321
121234323 121234343 121234345 121234543 121234545 121234565 121234567 123210101 123210121
123210123 123212101 123212121 123212123 123212321 123212323 123212343 123212345 123232101
123232121 123232123 123232321 123232323 123232343 123232345 123234321 123234323 123234343
123234345 123234543 123234545 123234565 123234567 123432101 123432121 123432123 123432321
123432323 123432343 123432345 123434321 123434323 123434343 123434345 123434543 123434545
123434565 123434567 123454321 123454323 123454343 123454345 123454543 123454545 123454565
123454567 123456543 123456545 123456565 123456567 123456765 123456767 123456787 123456789
```

Ethiopian multiplication

```
func double (n) { n << 1 }
func halve  (n) { n >> 1 }
func isEven (n) { n&1 == 0 }

func ethiopian_mult(a, b) {
    var r = 0
    while (a > 0) {
        r += b if !isEven(a)
        a = halve(a)
        b = double(b)
    }
    return r
}

say ethiopian_mult(17, 34)
```

Output:

```
578
```

Euclid-Mullin sequence

```
func f(n) is cached {
    return 2 if (n == 1)
    lpf(1 + prod(1..^n, {|k| f(k) }))
}

say f.map(1..16)
say f.map(17..27)
```

Output:

```
[2, 3, 7, 43, 13, 53, 5, 6221671, 38709183810571, 139, 2801, 11, 17, 5471, 52662739, 23003]
[30693651606209, 37, 1741, 1313797957, 887, 71, 7127, 109, 23, 97, 159227]
```

Euler's constant 0.5772...

Built-in:

```
# 100 decimals of precision
local Num!PREC = 4*100
say Num.EulerGamma
```

Output:

```
0.57721566490153286060651209008240243104215933593992359880576723488486772677766467093694706329174674951
```

Several formulas:

```
const n = (ARGV ? Num(ARGV[0]) : 50)       # number of iterations

define ℯ = Num.e
define π = Num.pi
define γ = Num.EulerGamma

func display(r, t) {
    say "#{r}\terror: #{ '%.0g' % abs(r - t) }"
}

# Original definition of the Euler-Mascheroni constant, due to Euler (1731)
display(sum(1..n, {|n| 1/n }) - log(n), γ)

# Formula due to Euler (best convergence)
display(harmfrac(n) - log(n) - 1/(2*n) - sum(1..n, {|k|
    -bernoulli(2*k) / (2*k) / n**(2*k)
}), γ)

# Formula derived from the above formula of Euler,
# using approximations of Bernoulli numbers.
display(harmfrac(n) - log(n) - 1/(2*n) - sum(1..n, {|k|
    (-1)**k * 4 * sqrt(π*k) * (π * ℯ)**(-2*k) * k**(2*k) / (2*k) / n**(2*k)
}), γ)

# Euler-Mascheroni constant, involving zeta(n)
display(1 - sum(2..(n+1), {|n|
    (zeta(n) - 1) / n
}), γ)

# Limit_{n->Infinity} zeta((n+1)/n) - n} = gamma
display(zeta((n+1)/n) - n, γ)

# Series due to Euler (1731).
display(sum(2..(n+1), {|n|
    (-1)**n * zeta(n) / n
}),  γ)

# Formula due to Euler in terms of log(2) and the odd zeta values
display(3/4 - log(2)/2 + sum(1..n, {|n|
    (1 - 1/(2*n + 1)) * (zeta(2*n + 1) - 1)
}), γ)

# Formula due to Euler in terms of log(2) and the odd zeta values (VII)
display(log(2) - sum(1..n, {|n|
    zeta(2*n + 1) / (2*n + 1) / 2**(2*n)
}), γ)

# Formula due to Vacca (1910)
display(sum(1..n, {|n|
    (-1)**n * floor(log2(n)) / n
}), γ)
```

Output:

```
0.587182332901278998941721005054217244843898981O7       error: 0.01
0.577215664901532860606512090082402431042159335 94       error: 2e-58
0.577201775274185974649592917416402750312879661543      error: 1e-05
0.577215664901532868991217643213156967011395887777      error: 8e-18
0.578670041015603217613302535407415749695401281 77       error: 0.001
0.567507841748305076772446986005418728718501189232      error: 0.01
0.577215664901532860606512090082272222693164663104      error: 1e-31
0.57721566490153286060651209008240496797924366087       error: 3e-33
0.611009392556929160631547597803236563977259982583      error: 0.03
```

Euler's identity

```
say ('e**iπ + 1 ≅ 0 : ', Num.e**Num.pi.i + 1 ≅ 0)
say ('Error: ', Num.e**Num.pi.i + 1)
```

Output:

```
e**iπ + 1 ≅ 0 : true
Error: -2.426619226245865820470287641579448361221225133 08e-58i
```

Euler's sum of powers conjecture

```
define range = (1 ..^ 250)

var p5 = Hash()
var sum2 = Hash()

for i in (range) {
    p5{i**5} = i
    for j in (range) {
        sum2{i**5 + j**5} = [i, j]
    }
}

var sk = sum2.keys.map{ Num(_) }.sort

for p in (p5.keys.map{ Num(_) }.sort) {

    var s = sk.first {|s|
        p > s && sum2.exists(p-s)
    } \\ next

    var t = (sum2{s} + sum2{p-s} -> map{|n| "#{n}⁵" }.join(' + '))
    say "#{t} = #{p5{p}}⁵"
    break
}
```

Output:

```
84⁵ + 27⁵ + 133⁵ + 110⁵ =  144⁵
```

Euler method

```
func euler_method(t0, t1, k, step_size) {
    var results = [[0, t0]]
    for s in (step_size..100 -> by(step_size)) {
        t0 -= ((t0 - t1) * k * step_size)
        results << [s, t0]
    }
    return results;
}

func analytical(t0, t1, k, time) {
    (t0 - t1) * exp(-time * k) + t1
}

var (T0, T1, k) = (100, 20, .07)
var r2  = euler_method(T0, T1, k,  2).grep { _[0] %% 10 }
var r5  = euler_method(T0, T1, k,  5).grep { _[0] %% 10 }
var r10 = euler_method(T0, T1, k, 10).grep { _[0] %% 10 }

say "Time\t      2     err(%)      5     err(%)    10      err(%)  Analytic"
say "-"*76

r2.range.each { |i|
    var an = analytical(T0, T1, k, r2[i][0])
    printf("%4d\t#{'%9.3f' * 7}\n",
                 r2[i][0],
                 r2[i][1], ( r2[i][1] / an) * 100 - 100,
                 r5[i][1], ( r5[i][1] / an) * 100 - 100,
                r10[i][1], (r10[i][1] / an) * 100 - 100,
                an)
}
```

Output:

```
Time          2     err(%)      5     err(%)    10      err(%)  Analytic
----------------------------------------------------------------------------
   0      100.000    0.000  100.000    0.000  100.000    0.000  100.000
  10       57.634   -3.504   53.800   -9.923   44.000  -26.331   59.727
  20       37.704   -5.094   34.281  -13.711   27.200  -31.534   39.728
  30       28.328   -4.927   26.034  -12.629   22.160  -25.629   29.797
  40       23.918   -3.808   22.549   -9.313   20.648  -16.959   24.865
  50       21.843   -2.555   21.077   -5.972   20.194   -9.910   22.416
  60       20.867   -1.569   20.455   -3.512   20.058   -5.384   21.200
  70       20.408   -0.912   20.192   -1.959   20.017   -2.808   20.596
  80       20.192   -0.512   20.081   -1.057   20.005   -1.432   20.296
  90       20.090   -0.281   20.034   -0.559   20.002   -0.721   20.147
 100       20.042   -0.152   20.014   -0.291   20.000   -0.361   20.073
```

Evaluate binomial coefficients

Straightforward translation of the formula:

```
func binomial(n,k) {
    n! / ((n-k)! * k!)
}

say binomial(400, 200)
```

Alternatively, by using the *Number.nok()* method:

```
say 400.nok(200)
```

Even or odd

Built-in methods:

```
var n = 42;
say n.is_odd;       # false
say n.is_even;      # true
```

Checking the last significant digit:

```
func is_odd(n)  { n&1 == 1 };
func is_even(n) { n&1 == 0 };
```

Using modular congruences:

```
func is_odd(n)  { n%2 == 1 };
func is_even(n) { n%2 == 0 };
```

Evolutionary algorithm

```
define target = "METHINKS IT IS LIKE A WEASEL"
define mutate_chance = 0.08
define alphabet = [@|('A'..'Z'), ' ']
define C = 100

func fitness(str) { str.chars ~Z== target.chars -> count(true) }
func mutate(str)  { str.gsub(/(.)/, {|s1| 1.rand < mutate_chance ? alphabet.pick : s1 }) }

for (
    var (i, parent) = (0, alphabet.rand(target.len).join);
    parent != target;
    parent = C.of{ mutate(parent) }.max_by(fitness)
) { printf("%6d: '%s'\n", i++, parent) }
```

Exactly three adjacent 3 in lists

```
func contains_n_consecutive_objs(arr, n, obj) {

    # In Sidef >= 3.99, we can also say:
    # arr.contains(n.of(obj)...)

    arr.each_cons(n, {|*a|
        if (a.all { _ == obj }) {
            return true
        }
    })

    return false
}

var lists = [
    [9,3,3,3,2,1,7,8,5],
    [5,2,9,3,3,7,8,4,1],
    [1,4,3,6,7,3,8,3,2],
    [1,2,3,4,5,6,7,8,9],
    [4,6,8,7,2,3,3,3,1],
]

lists.each {|list|
    say (list, " => ", contains_n_consecutive_objs(list, 3, 3))
}
```

Output:

```
[9, 3, 3, 3, 2, 1, 7, 8, 5] => true
[5, 2, 9, 3, 3, 7, 8, 4, 1] => false
[1, 4, 3, 6, 7, 3, 8, 3, 2] => false
[1, 2, 3, 4, 5, 6, 7, 8, 9] => false
[4, 6, 8, 7, 2, 3, 3, 3, 1] => true
```

Exceptions

An exception is thrown by the *die* keyword, which, if not caught, it terminates the program with an appropriate exit code.

```
try  {
    die "I'm dead!"        # throws an exception
}
catch { |msg|
    say "msg: #{msg}"      # msg: I'm dead! at test.sf line 2.
}

say "I'm alive..."
die "Now I'm dead!"        # this line terminates the program
say "Or am I?"             # Yes, you are!
```

Output:

```
msg: I'm dead! at test.sf line 2.
I'm alive...
Now I'm dead! at test.sf line 9.
```

Exceptions/Catch an exception thrown in a nested call

```
func baz(i) { die "U#{i}" }
func bar(i) { baz(i)      }

func foo {
    [0, 1].each { |i|
        try   { bar(i) }
        catch { |msg|
            msg ~~ /^U0/ ? say "Function foo() caught exception U0"
                         : die msg       # re-raise the exception
        }
    }
}

foo()
```

Output:

```
Function foo() caught exception U0
U1 at test.sf line 1. at test.sf line 9.
```

Executable library

Library saved as *Hailstone.sm*

```
func hailstone(n) {
    gather {
        while (n > 1) {
            take(n)
            n = (n.is_even ? n/2 : (3*n + 1))
        }
        take(1)
    }
}

if (__FILE__ == __MAIN__) {             # true when not imported
    var seq = hailstone(27)
    say "hailstone(27) - #{seq.len} elements: #{seq.ft(0, 3)} [...] #{seq.ft(-4)}"

    var n = 0
    var max = 0
    100_000.times { |i|
        var seq = hailstone(i)
        if (seq.len > max) {
            max = seq.len
            n = i
        }
    }

    say "Longest sequence is for #{n}: #{max}"
}
```

It can be run with:

```
$ sidef Hailstone.sm
```

It can then be used with a program such as:

```
include Hailstone

var score = Hash()
100_000.times { |i| score{ Hailstone::hailstone(i).len } := 0 ++ }

var k = score.keys.max_by {|k| score{k} }
say "Most common length is #{k}, occurring #{score{k}} times"
```

Called with a command line as:

```
$ sidef test_hailstone.sf
```

The library is searched in the directories specified in the *SIDEF_INC* environment variable, which defaults to the current directory.

Execute a system command

```
# Pipe in read-only mode
%p(ls).open_r.each { |line|
    print line;
};

var str1 = `ls`;         # backtick: returns a string
var str2 = %x(ls);       # ditto, alternative syntax

Sys.system('ls');   # system: executes a command and prints the result
Sys.exec('ls');     # replaces current process with another
```

Execute Brain****

```
define tape_length = 50_000;
define eof_val = -1;
define unbalanced_exit_code = 1;

var cmd = 0;
var cell = 0;
var code = [];
var loops = [];
var tape = tape_length.of(0);

func get_input {
    static input_buffer = [];
    input_buffer.len || (input_buffer = ((STDIN.readline \\ return eof_val).chomp.chars.map{.ord}));
    input_buffer.shift \\ eof_val;
}

func jump {
    var depth = 0;
    while (depth >= 0) {
        ++cmd < code.len || Sys.exit(unbalanced_exit_code);
        if (code[cmd] == '[') {
            ++depth;
        }
        elsif (code[cmd] == ']') {
            --depth;
        }
    }
}

var commands = Hash.new(
    '>' => { ++cell },
    '<' => { --cell },
    '+' => { ++tape[cell] },
    '-' => { --tape[cell] },
    '.' => { tape[cell].chr.print },
    ',' => { tape[cell] = get_input() },
    '[' => { tape[cell] ? loops.append(cmd) : jump() },
    ']' => { cmd = (loops.pop - 1) },
);

STDOUT.autoflush(1);
code = ARGF.slurp.chars.grep {|c| commands.exists(c)};
var code_len = code.len;

while (cmd < code_len) {
    commands{code[cmd]}.run;
    cmd++;
}
```

Execute HQ9+

```
class HQ9Interpreter {
    has pointer;
    has accumulator;

    func bob (beer) {
        func what  { "#{beer ? beer : 'No more'} bottle#{beer-1 ? 's' : ''} of beer" }
        func where { 'on the wall' }
        func drink { beer--; "Take one down, pass it around," }

        while (beer.is_pos) {
            [[what(), where()], [what()],
            [drink()], [what(), where()], []].each{.join(' ').say}
        }
    }

    method run (code) {
        var chars = code.chars;
        accumulator = 0;
        pointer = 0;
        while (pointer < chars.len) {
            given (chars[pointer].lc) {
                when ('h') { say 'Hello world!' }
                when ('q') { say code }
                when ('9') { bob(99) }
                when ('+') { accumulator++ }
                default    { warn %Q(Syntax error: Unknown command "#{chars[pointer]}") }
            }
            pointer++;
        }
    }
}
```

Usage:

```
var hq9 = HQ9Interpreter();
hq9.run("hHq+++Qq");
```

Output:

```
Hello world!
Hello world!
hHq+++Qq
hHq+++Qq
hHq+++Qq
```

Or start a REPL (Read Execute Print Loop) and interact at the command line:

```
var hq9 = HQ9Interpreter();
loop {
    var in = read('HQ9+>', String) \\ break;
    hq9.run(in)
}
```

Exponentiation operator

Function definition:

```
func expon(_, { .is_zero }) { 1 }

func expon(base, exp { .is_neg }) {
    expon(1/base, -exp)
}

func expon(base, exp { .is_int }) {

  var c = 1
  while (exp > 1) {
    c *= base if exp.is_odd
    base *= base
    exp >>= 1
  }

  return (base * c)
}

say expon(3, 10)
say expon(5.5, -3)
```

Operator definition:

```
class Number {
    method ⊙(exp) {
        expon(self, exp)
    }
}

say (3 ⊙ 10)
say (5.5 ⊙ -3)
```

Output:

```
59049
0.006010518407212622088655146506386175 81
```

Exponentiation order

In Sidef, the whitespace between the operands and the operator controls the precedence of the operation.

```
var a = [
    '5**3**2',
    '(5**3)**2',
    '5**(3**2)',
    '5 ** 3 ** 2',
    '5 ** 3**2',
    '5**3 ** 2',
    '[5,3,2]«**»',
]

a.each {|e|
    "%-12s == %s\n".printf(e, eval(e))
}
```

Output:

```
5**3**2      == 1953125
(5**3)**2    == 15625
5**(3**2)    == 1953125
5 ** 3 ** 2  == 15625
5 ** 3**2    == 1953125
5**3 ** 2    == 15625
[5,3,2]«**»  == 15625
```

Extend your language

```
class if2(cond1, cond2) {
    method then(block) {    # both true
        if (cond1 && cond2) {
            block.run;
        }
        return self;
    }
    method else1(block) {   # first true
        if (cond1 && !cond2) {
            block.run;
        }
        return self;
    }
    method else2(block) {   # second true
        if (cond2 && !cond1) {
            block.run;
        }
        return self;
    }
    method else(block) {    # none true
        if (!cond1 && !cond2) {
            block.run;
        }
        return self;
    }
}

if2(false, true).then {
    say "if2";
}.else1 {
    say "else1";
}.else2 {
    say "else2";        # <- this gets printed
}.else {
    say "else"
}
```

Extensible prime generator

```
say ("First 20: ", 20.nth_prime.primes.join(' '))
say ("Between 100 and 150: ", primes(100,150).join(' '))
say (prime_count(7700,8000), " primes between 7700 and 8000")
say ("10,000th prime: ", nth_prime(10_000))
```

Output:

```
First 20: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
Between 100 and 150: 101 103 107 109 113 127 131 137 139 149
30 primes between 7700 and 8000
10,000th prime: 104729
```

Extra primes

Simple solution:

```
say 1e4.primes.grep { .digits.all { .is_prime } && .sumdigits.is_prime }
```

Output:

```
[2, 3, 5, 7, 23, 223, 227, 337, 353, 373, 557, 577, 733, 757, 773, 2333, 2357, 2377, 2557, 2753, 2777,
```

Generate such primes from digits (faster):

```
func extra_primes(upto, base = 10) {

    upto = prev_prime(upto+1)

    var list = []
    var digits = @(^base)

    var prime_digits = digits.grep { .is_prime }
    var end_digits   = prime_digits.grep { .is_coprime(base) }

    list << prime_digits.grep { !.is_coprime(base) }...

    for k in (0 .. upto.ilog(base)) {
        prime_digits.variations_with_repetition(k, {|*a|
            next if ([end_digits[0], a...].digits2num(base) > upto)
            end_digits.each {|d|
                var n = [d, a...].digits2num(base)
                list << n if (n.is_prime && n.sumdigits(base).is_prime)
            }
        })
    }

    list.sort
}

with (1e4) { |n|
    say "Extra primes <= #{n.commify}:"
    say extra_primes(n).join(' ')
}

with (1000000000) {|n|
    say "\nLast 10 extra primes <= #{n.commify}:"
    say extra_primes(n).last(10).join(' ')
}
```

Output:

```
Extra primes <= 10,000:
2 3 5 7 23 223 227 337 353 373 557 577 733 757 773 2333 2357 2377 2557 2753 2777 3253 3257 3323 3527 37

Last 10 extra primes <= 1,000,000,000:
777753773 777755753 777773333 777773753 777775373 777775553 777775577 777777227 777777577 777777773
```

Extract file extension

```
func extension(filename) {
    filename.match(/(\.[a-z0-9]+)\z/i).to_s
}

var files = [
    'http://example.com/download.tar.gz',
    'CharacterModel.3DS',
    '.desktop',
    'document',
    'document.txt_backup',
    '/etc/pam.d/login',
]

files.each {|f|
    printf("%-36s -> %-11s\n", f.dump, extension(f).dump)
}
```

Output:

```
"http://example.com/download.tar.gz" -> ".gz"
"CharacterModel.3DS"                 -> ".3DS"
".desktop"                           -> ".desktop"
"document"                           -> ""
"document.txt_backup"                -> ""
"/etc/pam.d/login"                   -> ""
```

Extreme floating point values

NaN and *Inf* literals can be used to represent the *Not-a-Number* and *Infinity* values, which are returned in special cases, such as *0/0* and *1/0*. However, one thing to notice, is that in Sidef there is no distinction between *0.0* and *-0.0* and can't be differentiated from each other.

```
var inf = 1/0    # same as: Inf
var nan = 0/0    # same as: NaN

var exprs = [
  "1.0 / 0.0", "-1.0 / 0.0", "0.0 / 0.0", "- 0.0",
  "inf + 1", "5 - inf", "inf * 5", "inf / 5", "inf * 0",
  "1.0 / inf", "-1.0 / inf", "inf + inf", "inf - inf",
  "inf * inf", "inf / inf", "inf * 0.0", " 0 < inf", "inf == inf",
  "nan + 1", "nan * 5", "nan - nan", "nan * inf", "- nan",
  "nan == nan", "nan > 0", "nan < 0", "nan == 0", "0.0 == -0.0",
]

exprs.each { |expr|
  "%15s => %s\n".printf(expr, eval(expr))
}

say "-"*40
say("NaN equality: ",        NaN ==  nan)
say("Infinity equality: ",   Inf ==  inf)
say("-Infinity equality: ", -Inf == -inf)

say "-"*40
say("sqrt(-1)   = ",   sqrt(-1))
say("tanh(-Inf) = ", tanh(-inf))
say("(-Inf)**2  = ",  (-inf)**2)
say("(-Inf)**3  = ",  (-inf)**3)
say("acos(Inf)  = ",  acos(inf))
say("atan(Inf)  = ",  atan(inf))
say("log(-1)    = ",    log(-1))
say("atanh(Inf) = ", atanh(inf))
```

Output:

```
      1.0 / 0.0 => Inf
     -1.0 / 0.0 => -Inf
      0.0 / 0.0 => NaN
          - 0.0 => 0
        inf + 1 => Inf
        5 - inf => -Inf
        inf * 5 => Inf
        inf / 5 => Inf
        inf * 0 => NaN
      1.0 / inf => 0
     -1.0 / inf => 0
      inf + inf => Inf
      inf - inf => NaN
      inf * inf => Inf
      inf / inf => NaN
      inf * 0.0 => NaN
        0 < inf => true
     inf == inf => true
        nan + 1 => NaN
        nan * 5 => NaN
      nan - nan => NaN
      nan * inf => NaN
          - nan => NaN
     nan == nan => false
        nan > 0 =>
        nan < 0 =>
       nan == 0 => false
    0.0 == -0.0 => true
----------------------------------------
NaN equality: false
Infinity equality: true
-Infinity equality: true
----------------------------------------
sqrt(-1)   = i
tanh(-Inf) = -1
(-Inf)**2  = Inf
(-Inf)**3  = -Inf
acos(Inf)  = -Infi
atan(Inf)  = 1.5707963267948966192313216916397514420985846996 9
log(-1)    = 3.1415926535897932384626433832795028841971693993 8i
atanh(Inf) = 1.5707963267948966192313216916397514420985846996 9i
```

Factorial

Recursive:

```
func factorial_recursive(n) {
    n == 0 ? 1 : (n * __FUNC__(n-1))
}
```

Catamorphism:

```
func factorial_reduce(n) {
    1..n -> reduce({|a,b| a * b }, 1)
}
```

Iterative:

```
func factorial_iterative(n) {
    var f = 1
    {|i| f *= i } << 2..n
    return f
}
```

Built-in:

```
say 5!
```

Factorions

```
func max_power(b = 10) {
    var m = 1
    var f = (b-1)!
    while (m*f >= b**(m-1)) {
        m += 1
    }
    return m-1
}

func factorions(b = 10) {

    var result = []
    var digits = @^b
    var fact = digits.map { _! }

    for k in (1 .. max_power(b)) {
        digits.combinations_with_repetition(k, {|*comb|
            var n = comb.sum_by { fact[_] }
            if (n.digits(b).sort == comb) {
                result << n
            }
        })
    }

    return result
}

for b in (2..12) {
    var r = factorions(b)
    say "Base #{'%2d' % b} factorions: #{r}"
}
```

Output:

```
Base  2 factorions: [1, 2]
Base  3 factorions: [1, 2]
Base  4 factorions: [1, 2, 7]
Base  5 factorions: [1, 2, 49]
Base  6 factorions: [1, 2, 25, 26]
Base  7 factorions: [1, 2]
Base  8 factorions: [1, 2]
Base  9 factorions: [1, 2, 41282]
Base 10 factorions: [1, 2, 145, 40585]
Base 11 factorions: [1, 2, 26, 48, 40472]
Base 12 factorions: [1, 2]
```

Factors of a Mersenne number

```
func mtest(b, p) {
    var bits = b.base(2).digits
    for (var sq = 1; bits; sq %= p) {
        sq *= sq
        sq += sq if bits.shift==1
    }
    sq == 1
}

for m (2..60 -> grep{ .is_prime }, 929) {
    var f = 0
    var x = (2**m - 1)
    var q
    { |k|
        q = (2*k*m + 1)
        q%8 ~~ [1,7] || q.is_prime || next
        q*q > x || (f = mtest(m, q)) && break
    } << 1..Inf
    say (f ? "M#{m} is composite with factor #{q}"
           : "M#{m} is prime")
}
```

Output:

```
M2 is prime
M3 is prime
M5 is prime
M7 is prime
M11 is composite with factor 23
M13 is prime
M17 is prime
M19 is prime
M23 is composite with factor 47
M29 is composite with factor 233
M31 is prime
M37 is composite with factor 223
M41 is composite with factor 13367
M43 is composite with factor 431
M47 is composite with factor 2351
M53 is composite with factor 6361
M59 is composite with factor 179951
M929 is composite with factor 13007
```

Factors of an integer

Built-in:

```
say divisors(97)    #=> [1, 97]
say divisors(2695)  #=> [1, 5, 7, 11, 35, 49, 55, 77, 245, 385, 539, 2695]
```

Trial division (slow for large n):

```
func divisors(n) {
  gather {
    { |d|
        take(d, n//d) if d.divides(n)
    } << 1..n.isqrt
  }.sort.uniq
}

[53, 64, 32766].each {|n|
    say "divisors(#{n}): #{divisors(n)}"
}
```

Output:

```
divisors(53): [1, 53]
divisors(64): [1, 2, 4, 8, 16, 32, 64]
divisors(32766): [1, 2, 3, 6, 43, 86, 127, 129, 254, 258, 381, 762, 5461, 10922, 16383, 32766]
```

Fairshare between two and more

```
for b in (2,3,5,11) {
    say ("#{'%2d' % b}: ", 25.of { .sumdigits(b) % b })
}
```

Output:

```
 2: [0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0]
 3: [0, 1, 2, 1, 2, 0, 2, 0, 1, 1, 2, 0, 2, 0, 1, 0, 1, 2, 2, 0, 1, 0, 1, 2, 1]
 5: [0, 1, 2, 3, 4, 1, 2, 3, 4, 0, 2, 3, 4, 0, 1, 3, 4, 0, 1, 2, 4, 0, 1, 2, 3]
11: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 2, 3, 4]
```

Farey sequence

```
func farey_count(n) {   # A005728
    1 + sum(1..n, {|k| euler_phi(k) })
}

func farey(n) {

    var seq = [0]
    var (a,b,c,d) = (0,1,1,n)

    while (c <= n) {
        var k = (n+b)//d
        (a,b,c,d) = (c, d, k*c - a, k*d - b)
        seq << a/b
    }

    return seq
}

say "Farey sequence for order 1 through 11 (inclusive):"
for n in (1..11) {
    say("F(%2d): %s" % (n, farey(n).map{.as_frac}.join(" ")))
}

say "\nNumber of fractions in the Farey sequence:"
for n in (100..1000 -> by(100)) {
    say ("F(%4d) =%7d" % (n, farey_count(n)))
}
```

Output:

```
Farey sequence for order 1 through 11 (inclusive):
F( 1): 0/1 1/1
F( 2): 0/1 1/2 1/1
F( 3): 0/1 1/3 1/2 2/3 1/1
F( 4): 0/1 1/4 1/3 1/2 2/3 3/4 1/1
F( 5): 0/1 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1/1
F( 6): 0/1 1/6 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 5/6 1/1
F( 7): 0/1 1/7 1/6 1/5 1/4 2/7 1/3 2/5 3/7 1/2 4/7 3/5 2/3 5/7 3/4 4/5 5/6 6/7 1/1
F( 8): 0/1 1/8 1/7 1/6 1/5 1/4 2/7 1/3 3/8 2/5 3/7 1/2 4/7 3/5 5/8 2/3 5/7 3/4 4/5 5/6 6/7 7/8 1/1
F( 9): 0/1 1/9 1/8 1/7 1/6 1/5 2/9 1/4 2/7 1/3 3/8 2/5 3/7 4/9 1/2 5/9 4/7 3/5 5/8 2/3 5/7 3/4 7/9 4/5
F(10): 0/1 1/10 1/9 1/8 1/7 1/6 1/5 2/9 1/4 2/7 3/10 1/3 3/8 2/5 3/7 4/9 1/2 5/9 4/7 3/5 5/8 2/3 7/10 5
F(11): 0/1 1/11 1/10 1/9 1/8 1/7 1/6 2/11 1/5 2/9 1/4 3/11 2/7 3/10 1/3 4/11 3/8 2/5 3/7 4/9 5/11 1/2 6

Number of fractions in the Farey sequence:
F( 100) =   3045
F( 200) =  12233
F( 300) =  27399
F( 400) =  48679
F( 500) =  76117
F( 600) = 109501
F( 700) = 149019
F( 800) = 194751
F( 900) = 246327
F(1000) = 304193
```

Fast Fourier transform

```
func fft(arr) {
    arr.len == 1 && return arr

    var evn = fft([arr[^arr -> grep { .is_even }]])
    var odd = fft([arr[^arr -> grep { .is_odd  }]])
    var twd = (Num.tau.i / arr.len)

    ^odd -> map {|n| odd[n] *= exp(twd * n) }
    (evn »+« odd) + (evn »-« odd)
}

var cycles = 3
var sequence = 0..15
var wave = sequence.map {|n| sin(n * Num.tau / sequence.len * cycles) }
say "wave:#{wave.map{|w| '%6.3f' % w }.join(' ')}"
say "fft: #{fft(wave).map { '%6.3f' % .abs }.join(' ')}"
```

Output:

```
wave: 0.000  0.924  0.707 -0.383 -1.000 -0.383  0.707  0.924  0.000 -0.924 -0.707  0.383  1.000  0.383
fft:  0.000  0.000  0.000  8.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  8.000
```

FASTA format

```
func fasta_format(strings) {
    var out = []
    var text = ''
    for line in (strings.lines) {
        if (line.begins_with('>')) {
            text.len && (out << text)
            text = line.substr(1)+': '
        }
        else {
            text += line
        }
    }
    text.len && (out << text)
    return out
}

fasta_format(DATA.slurp).each { .say }

__DATA__
>Rosetta_Example_1
THERECANBENOSPACE
>Rosetta_Example_2
THERECANBESEVERAL
LINESBUTTHEYALLMUST
BECONCATENATED
```

Output:

```
Rosetta_Example_1: THERECANBENOSPACE
Rosetta_Example_2: THERECANBESEVERALLINESBUTTHEYALLMUSTBECONCATENATED
```

Faulhaber's formula

```
const AnyNum = require('Math::AnyNum')
const Poly   = require('Math::Polynomial')

Poly.string_config(Hash(
    fold_sign => true, prefix => "",
    suffix => "", variable => "n"
))

func anynum(n) {
    AnyNum.new(n.as_rat)
}

func faulhaber_formula(p) {
    (p+1).of { |j|
        Poly.monomial(p - j + 1)\
            .mul_const(anynum(bernoulli(j)))\
            .mul_const(anynum(binomial(p+1, j)))
    }.reduce(:add).div_const(p+1)
}

for p in (^10) {
    printf("%2d: %s\n", p, faulhaber_formula(p))
}
```

Output:

```
 0: n
 1: 1/2 n^2 + 1/2 n
 2: 1/3 n^3 + 1/2 n^2 + 1/6 n
 3: 1/4 n^4 + 1/2 n^3 + 1/4 n^2
 4: 1/5 n^5 + 1/2 n^4 + 1/3 n^3 - 1/30 n
 5: 1/6 n^6 + 1/2 n^5 + 5/12 n^4 - 1/12 n^2
 6: 1/7 n^7 + 1/2 n^6 + 1/2 n^5 - 1/6 n^3 + 1/42 n
 7: 1/8 n^8 + 1/2 n^7 + 7/12 n^6 - 7/24 n^4 + 1/12 n^2
 8: 1/9 n^9 + 1/2 n^8 + 2/3 n^7 - 7/15 n^5 + 2/9 n^3 - 1/30 n
 9: 1/10 n^10 + 1/2 n^9 + 3/4 n^8 - 7/10 n^6 + 1/2 n^4 - 3/20 n^2
```

Faulhaber's triangle

```
func faulhaber_triangle(p) {
    { binomial(p, _) * bernoulli(_) / p }.map(p ^.. 0)
}

{ |p|
    say faulhaber_triangle(p).map{ '%6s' % .as_rat }.join
} << 1..10

const p = 17
const n = 1000

say ''
say faulhaber_triangle(p+1).map_kv {|k,v| v * n**(k+1) }.sum
```

Output:

```
     1
   1/2   1/2
   1/6   1/2   1/3
     0   1/4   1/2   1/4
 -1/30     0   1/3   1/2   1/5
     0 -1/12     0  5/12   1/2   1/6
  1/42     0  -1/6     0   1/2   1/2   1/7
     0  1/12     0 -7/24     0  7/12   1/2   1/8
 -1/30     0   2/9     0 -7/15     0   2/3   1/2   1/9
     0 -3/20     0   1/2     0 -7/10     0   3/4   1/2  1/10

560569722165555801110300779619441834001983332730500000
```

Alternative solution:

```
func find_poly_degree(a) {
    var c = 0
    loop {
        ++c
        a = a.map_cons(2, {|n,k| n-k })
        return 0 if a.is_empty
        return c if a.all { .is_zero }
    }
}

func faulhaber_triangle(n) {
    var a = (0..(n+2) -> accumulate { _**n })
    var c = find_poly_degree(a)

    var A = c.of {|n|
        c.of {|k| n**k }
    }

    A.msolve(a).slice(1)
}

10.times { say faulhaber_triangle(_) }
```

Output:

```
[1]
[1/2, 1/2]
[1/6, 1/2, 1/3]
[0, 1/4, 1/2, 1/4]
[-1/30, 0, 1/3, 1/2, 1/5]
[0, -1/12, 0, 5/12, 1/2, 1/6]
[1/42, 0, -1/6, 0, 1/2, 1/2, 1/7]
[0, 1/12, 0, -7/24, 0, 7/12, 1/2, 1/8]
[-1/30, 0, 2/9, 0, -7/15, 0, 2/3, 1/2, 1/9]
[0, -3/20, 0, 1/2, 0, -7/10, 0, 3/4, 1/2, 1/10]
```

Feigenbaum constant calculation

```
var a1 = 1
var a2 = 0
var δ  = 3.2.float

say " i\tδ"

for i in (2..15) {
    var a0 = ((a1 - a2)/δ + a1)
    10.times {
        var (x, y) = (0, 0)
        2**i -> times {
            y = (1 - 2*x*y)
            x = (a0 - x²)
        }
        a0 -= x/y
    }
    δ = ((a1 - a2) / (a0 - a1))
    (a2, a1) = (a1, a0)
    printf("%2d %.8f\n", i, δ)
}
```

Output:

```
 i      δ
 2 3.21851142
 3 4.38567760
 4 4.60094928
 5 4.65513050
 6 4.66611195
 7 4.66854858
 8 4.66906066
 9 4.66917156
10 4.66919516
11 4.66920023
12 4.66920131
13 4.66920155
14 4.66920160
15 4.66920161
```

Fermat numbers

```
func fermat_number(n) {
    2**(2**n) + 1
}

func fermat_one_factor(n) {
    fermat_number(n).ecm_factor
}

for n in (0..9) {
    say "F_#{n} = #{fermat_number(n)}"
}

say ''

for n in (0..13) {
    var f = fermat_one_factor(n)
    say ("F_#{n} = ", join(' * ', f.shift,
      f.map { <C P>[.is_prime] + .len }...))
}
```

Output:

```
F_0 = 3
F_1 = 5
F_2 = 17
F_3 = 257
F_4 = 65537
F_5 = 4294967297
F_6 = 18446744073709551617
F_7 = 340282366920938463463374607431768211457
F_8 = 115792089237316195423570985008687907853269984665640564039457584007913129639937
F_9 = 13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903

F_0 = 3
F_1 = 5
F_2 = 17
F_3 = 257
F_4 = 65537
F_5 = 641 * P7
F_6 = 274177 * P14
F_7 = 59649589127497217 * P22
F_8 = 1238926361552897 * P62
F_9 = 2424833 * C148
F_10 = 45592577 * C301
F_11 = 319489 * C612
F_12 = 114689 * C1228
F_13 = 2710954639361 * C2454
```

Fibonacci matrix-exponentiation

Computing the n-th Fibonacci number, using matrix-exponentiation (this function is also built-in):

```
func fibonacci(n) {
    ([[1,1],[1,0]]**n)[0][1]
}

say 15.of(fibonacci)    #=> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

First and last 20 digits of the n-th Fibonacci number:

```
for n in (16, 32) {

    var f = fibonacci(2**n)

    with (20) {|k|
        var a = (f // 10**(f.ilog10 - k + 1))
        var b = (f % 10**k)
        say "F(2^#{n}) = #{a} ... #{b}"
    }
}
```

Output:

```
F(2^16) = 73199214460290552832 ... 97270190955307463227
F(2^32) = 61999319689381859818 ... 39623735538208076347
```

More efficient approach, using Binet's formula for computing the first k digits, combined with the built-in method **fibmod(n,m)** for computing the last k digits:

```
func binet_approx(n) {
    const PHI =  (1.25.sqrt + 0.5)
    const IHP = -(1.25.sqrt - 0.5)
    (log(PHI)*n - log(PHI-IHP))
}

func nth_fib_first_k_digits(n,k) {
    var f = binet_approx(n)
    floor(exp(f - log(10)*(floor(f / log(10) + 1))) * 10**k)
}

func nth_fib_last_k_digits(n,k) {
    fibmod(n, 10**k)
}

for n in (16, 32, 64) {
    var first20 = nth_fib_first_k_digits(2**n, 20)
    var last20  = nth_fib_last_k_digits(2**n, 20)
    say "F(2^#{n}) = #{first20} ... #{last20}"
}
```

Output:

```
F(2^16) = 73199214460290552832 ... 97270190955307463227
F(2^32) = 61999319689381859818 ... 39623735538208076347
F(2^64) = 11175807536929528424 ... 17529800348089840187
```

Fibonacci n-step number sequences

```
func fib(n, xs=[1], k=20) {
    loop {
        var len = xs.len
        len >= k && break
        xs << xs.ft(max(0, len - n)).sum
    }
    return xs
}

for i in (2..10) {
    say fib(i).join(' ')
}
say fib(2, [2, 1]).join(' ')
```

Output:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768 10609 19513 35890 66012
1 1 2 4 8 15 29 56 108 208 401 773 1490 2872 5536 10671 20569 39648 76424 147312
1 1 2 4 8 16 31 61 120 236 464 912 1793 3525 6930 13624 26784 52656 103519 203513
1 1 2 4 8 16 32 63 125 248 492 976 1936 3840 7617 15109 29970 59448 117920 233904
1 1 2 4 8 16 32 64 127 253 504 1004 2000 3984 7936 15808 31489 62725 124946 248888
1 1 2 4 8 16 32 64 128 255 509 1016 2028 4048 8080 16128 32192 64256 128257 256005
1 1 2 4 8 16 32 64 128 256 511 1021 2040 4076 8144 16272 32512 64960 129792 259328
1 1 2 4 8 16 32 64 128 256 512 1023 2045 4088 8172 16336 32656 65280 130496 260864
2 1 3 4 7 11 18 29 47 76 123 199 322 521 843 1364 2207 3571 5778 9349
```

Using matrix exponentiation:

```
func fibonacci_matrix(k) is cached {
    Matrix.build(k,k, {|i,j|
        ((i == k-1) || (i == j-1)) ? 1 : 0
    })
}

func fibonacci_kth_order(n, k=2) {
    var A = fibonacci_matrix(k)
    (A**n)[0][-1]
}

for k in (2..9) {
    say ("Fibonacci of k=#{k} order: ", (15+k).of { fibonacci_kth_order(_, k) })
}
```

Output:

```
Fibonacci of k=2 order: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
Fibonacci of k=3 order: [0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, 927, 1705, 3136, 5768]
Fibonacci of k=4 order: [0, 0, 0, 1, 1, 2, 4, 8, 15, 29, 56, 108, 208, 401, 773, 1490, 2872, 5536, 1067
Fibonacci of k=5 order: [0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 31, 61, 120, 236, 464, 912, 1793, 3525, 6930, 1
Fibonacci of k=6 order: [0, 0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 32, 63, 125, 248, 492, 976, 1936, 3840, 7617
Fibonacci of k=7 order: [0, 0, 0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 32, 64, 127, 253, 504, 1004, 2000, 3984,
Fibonacci of k=8 order: [0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 32, 64, 128, 255, 509, 1016, 2028, 404
Fibonacci of k=9 order: [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 4, 8, 16, 32, 64, 128, 256, 511, 1021, 2040,
```

Faster algorithm:

```
func fibonacci_kth_order (n, k = 2) {

    return 0 if (n < k-1)

    var f = (1..(k+1) -> map {|j|
        j < k ? 2**j : 1
    })

    k += 1

    for i in (2*(k-1) .. n) {
        f[i%k] = (2*f[(i-1)%k] - f[i%k])
    }

    return f[n%k]
}

for k in (2..9) {
    say ("Fibonacci of k=#{k} order: ", (15+k).of { fibonacci_kth_order(_, k) })
}
```

(same output as above)

Fibonacci sequence

Iterative:

```
func fib_iter(n) {
    var (a, b) = (0, 1)
    { (a, b) = (b, a+b) } * n
    return a
}
```

Recursive:

```
func fib_rec(n) {
    n < 2 ? n : (__FUNC__(n-1) + __FUNC__(n-2))
}
```

Recursive with memoization:

```
func fib_mem (n) is cached {
    n < 2 ? n : (__FUNC__(n-1) + __FUNC__(n-2))
}
```

Closed-form:

```
func fib_closed(n) {
    define S = (1.25.sqrt + 0.5)
    define T = (-S + 1)
    (S**n - T**n) / (-T + S) -> round
}
```

Built-in:

```
say fib(12)        #=> 144
```

Fibonacci word

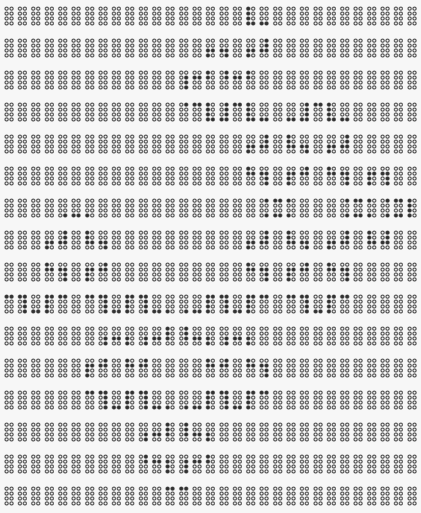
```
func entropy(s) {
    [0] + (s.chars.freq.values »/» s.len) -> reduce { |a,b|
        a - b*b.log2
    }
}

var n_max = 37
var words = ['1', '0']

{
    words.append(words[-1] + words[-2])
} * (n_max - words.len)

say ('%3s %10s %15s  %s' % <N Length Entropy Fibword>...)

for i in ^words {
    var word = words[i]
    say ('%3i %10i %15.12f  %s' % (i+1,
                                   word.len,
                                   entropy(word),
                                   word.len<30 ? word : '<too long>'))
}
```

Fibonacci word/fractal

```
var(m=17, scale=3) = ARGV.map{.to_i}...

(var world = Hash.new){0}{0} = 1
var loc = 0
var dir = 1i

var fib = ['1', '0']
func fib_word(n) {
    fib[n] \\= (fib_word(n-1) + fib_word(n-2))
}

func step {
    scale.times {
        loc += dir
        world{loc.im}{loc.re} = 1
    }
}

func turn_left  { dir *=  1i }
func turn_right { dir *= -1i }

var n = 1
fib_word(m).each { |c|
    if (c == '0') {
        step()
        n % 2 == 0 ? turn_left()
                   : turn_right()
    } else { n++ }
}

func braille_graphics(a) {
    var (xlo, xhi, ylo, yhi) = ([Inf, -Inf]*2)...

    a.each_key { |y|
        ylo.min!(y.to_i)
        yhi.max!(y.to_i)
        a{y}.each_key { |x|
            xlo.min!(x.to_i)
            xhi.max!(x.to_i)
        }
    }

    for y in (ylo..yhi `by` 4) {
        for x in (xlo..xhi `by` 2) {
            var cell = 0x2800

            a{y+0}{x+0} && (cell += 1)
            a{y+1}{x+0} && (cell += 2)
            a{y+2}{x+0} && (cell += 4)
            a{y+0}{x+1} && (cell += 8)
            a{y+1}{x+1} && (cell += 16)
            a{y+2}{x+1} && (cell += 32)
            a{y+3}{x+0} && (cell += 64)
            a{y+3}{x+1} && (cell += 128)

            print cell.chr
        }
        print "\n"
    }
}

braille_graphics(world)
```

Output:

```
$ sidef fib_word_fractal.sf 12 3
```



File extension is in extensions list

```
func check_extension(filename, extensions) {
    filename ~~ Regex('\.(' + extensions.map { .escape }.join('|') + ')\z', :i)
}
```

Testing:

```
var extensions = ['zip', 'rar', '7z', 'gz', 'archive', 'A##', 'tar.bz2']

var files = [
    'MyData.a##', 'MyData.tar.Gz', 'MyData.gzip', 'MyData.7z.backup',
    'MyData...', 'MyData', 'MyData_v1.0.tar.bz2', 'MyData_v1.0.bz2'
]

for file in files {
    printf("%-19s - %s\n", file, check_extension(file, extensions))
}
```

Output:

```
MyData.a##          - true
MyData.tar.Gz       - true
MyData.gzip         - false
MyData.7z.backup    - false
MyData...           - false
MyData              - false
MyData_v1.0.tar.bz2 - true
MyData_v1.0.bz2     - false
```

File input/output

```
var in = %f'input.txt'.open_r;
var out = %f'output.txt'.open_w;

in.each { |line|
    out.print(line);
};
```

File modification time

```
var file = File(__FILE__)
say file.stat.mtime            # seconds since the epoch

# keep atime unchanged
# set mtime to current time
file.utime(file.stat.atime, Time.now)
```

File size

```
say (Dir.cwd  + %f'input.txt' -> size)
say (Dir.root + %f'input.txt' -> size)
```

File size distribution

```
func traverse(Block callback, Dir dir) {
    dir.open(\var dir_h) || return nil

    for entry in (dir_h.entries) {
        if (entry.kind_of(Dir)) {
            traverse(callback, entry)
        } else {
            callback(entry)
        }
    }
}

var dir = (ARGV ? Dir(ARGV[0]) : Dir.cwd)

var group = Hash()
var files_num = 0
var total_size = 0

traverse({ |file|
    group{file.size+1 -> log10.round} := 0 += 1
    total_size += file.size
    files_num += 1
}, dir)

for k,v in (group.sort_by { |k,_| Num(k) }) {
    say "log10(size) ~~ #{k} -> #{v} files"
}

say "Total: #{total_size} bytes in #{files_num} files"
```

Output:

```
$ sidef script.sf /usr/bin
log10(size) ~~ 1 -> 4 files
log10(size) ~~ 2 -> 70 files
log10(size) ~~ 3 -> 246 files
log10(size) ~~ 4 -> 1337 files
log10(size) ~~ 5 -> 815 files
log10(size) ~~ 6 -> 167 files
log10(size) ~~ 7 -> 9 files
log10(size) ~~ 8 -> 2 files
Total: 370026462 bytes in 2650 files
```

Filter

```
var arr = [1,2,3,4,5]

# Creates a new array
var new = arr.grep {|i| i.is_even }
say new    # => [2, 4]

# Destructive (at variable level)
arr.grep! {|i| i.is_even }
say arr    # => [2, 4]
```

Find adjacent primes which differ by a square integer

```
var p = 2
var upto = 1e6

each_prime(p.next_prime, upto, {|q|
    if (q-p > 36 && is_square(q-p)) {
        say "#{'%6s' % q} - #{'%6s' % p} = #{'%2s' % isqrt(q-p)}^2"
    }
    p = q
})
```

Output:

```
 89753 -  89689 =  8^2
107441 - 107377 =  8^2
288647 - 288583 =  8^2
368021 - 367957 =  8^2
381167 - 381103 =  8^2
396833 - 396733 = 10^2
400823 - 400759 =  8^2
445427 - 445363 =  8^2
623171 - 623107 =  8^2
625763 - 625699 =  8^2
637067 - 637003 =  8^2
710777 - 710713 =  8^2
725273 - 725209 =  8^2
779477 - 779413 =  8^2
801947 - 801883 =  8^2
803813 - 803749 =  8^2
821741 - 821677 =  8^2
832583 - 832519 =  8^2
838349 - 838249 = 10^2
844841 - 844777 =  8^2
883871 - 883807 =  8^2
912167 - 912103 =  8^2
919511 - 919447 =  8^2
954827 - 954763 =  8^2
981887 - 981823 =  8^2
997877 - 997813 =  8^2
```

Find common directory path

```
var dirs = %w(
    /home/user1/tmp/coverage/test
    /home/user1/tmp/covert/operator
    /home/user1/tmp/coven/members
);

var unique_pref = dirs.map{.split('/')}.abbrev.min_by{.len};
var common_dir  = [unique_pref, unique_pref.pop][0].join('/');
say common_dir;   # => /home/user1/tmp
```

Find duplicate files

It uses the portable *File::Find* module which means that it should work, virtually, on any platform.

```
# usage: sidef fdf.sf [size] [dir1] [...]

require('File::Find')

func find_duplicate_files(Block code, size_min=0, *dirs) {
    var files = Hash()
    %S<File::Find>.find(
        Hash(
            no_chdir => true,
            wanted   => func(arg) {
                var file = File(arg)
                file.is_file || return()
                file.is_link && return()
                var size = file.size
                size >= size_min || return()
                files{size} := [] << file
            },
        ) => dirs...
    )

    files.values.each { |set|
        set.len > 1 || next
        var dups = Hash()
        for i in (^set.end) {
            for (var j = set.end; j > i; --j) {
                if (set[i].compare(set[j]) == 0) {
                    dups{set[i]} := [] << set.pop_at(j++)
                }
            }
        }
        dups.each{ |k,v| code(k.to_file, v...) }
    }

    return()
}

var duplicates = Hash()
func collect(*files) {
    duplicates{files[0].size} := [] << files
}

find_duplicate_files(collect, Num(ARGV.shift), ARGV...)

for k,v in (duplicates.sort_by { |k| -k.to_i }) {
    say "=> Size: #{k}\n#{'~'*80}"
    for files in v {
        say "#{files.sort.join(%Q[\n])}\n#{'-'*80}"
    }
}
```

Section of sample output:

Output:

```
% sidef fdf.sf 0 /tmp /usr/bin
=> Size: 5656
~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/bin/precat
/usr/bin/preunzip
/usr/bin/prezip
-------------------------
=> Size: 2305
~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/bin/gunzip
/usr/bin/uncompress
-------------------------
=> Size: 2
~~~~~~~~~~~~~~~~~~~~~~~~~
/tmp/a.txt
/tmp/b.txt
-------------------------
/tmp/m.txt
/tmp/n.txt
-------------------------
```

Find first and last set bit of a long integer

Sidef has arbitrary sized integers.

```
func msb(n) {
    var b = 0
    while(n >>= 1) { ++b }
    return b
}

func lsb(n) {
    msb(n & -n)
}
```

Test cases:

```
func table (base,power) {
    var digits = length(base**power)
    printf("%#{digits}s  lsb msb\n", 'number')
    for n in (0..power) {
        var x = base**n
        printf("%#{digits}s  %2s  %3s\n", x, lsb(x), msb(x))
    }
}

table(42, 20)
table(1302, 20)
```

Find largest left truncatable prime in a given base

```
func lltp(n) {
    var b = 1
    var best = nil
    var v = (n-1 -> primes)

    while (v) {
        best = v.max
        b *= n
        v.map! { |vi|
            {|i| i*b + vi }.map(1..^n).grep{.is_prime}...
        }
    }

    return best
}

for i in (3..17) {
    printf("%2d %s\n", i, lltp(i))
}
```

Output:

```
 3 23
 4 4091
 5 7817
 6 4836525320399
 7 817337
 8 14005650767869
 9 1676456897
10 357686312646216567629137
11 2276005673
12 1309243064773619081730313006582753 9
13 812751503
14 615419590422100474355767356763
15 340686457059276624472861 91
16 108830370715352164496834555998 7
17 13563641583101
```

Alternative solution:

```
func digits2num(digits, base) {
    digits.map_kv {|k,v| base**k * v  }.sum
}

func generate_from_suffix(p, base) {

    var seq = [p]

    for n in (1 ..^ base) {
        var t = [p..., n]
        if (is_prime(digits2num(t, base))) {
            seq << __FUNC__(t, base)...
        }
    }

    return seq
}

func left_truncatable_primes(base) {

    var prime_digits = (base-1 -> primes)

    prime_digits.map  {|p| generate_from_suffix([p], base)... }\
                .map  {|t| digits2num(t, base) }\
                .sort
}

for n in (3..11) {
    var ltp = left_truncatable_primes(n)
    say ("There are #{'%4d' % ltp.len} left-truncatable primes in base #{'%2d' % n}, where largest is
#{ltp.max}")
}
```

Output:

```
There are    3 left-truncatable primes in base  3, where largest is 23
There are   16 left-truncatable primes in base  4, where largest is 4091
There are   15 left-truncatable primes in base  5, where largest is 7817
There are  454 left-truncatable primes in base  6, where largest is 4836525320399
There are   22 left-truncatable primes in base  7, where largest is 817337
There are  446 left-truncatable primes in base  8, where largest is 14005650767869
There are  108 left-truncatable primes in base  9, where largest is 1676456897
There are 4260 left-truncatable primes in base 10, where largest is 357686312646216567629137
There are   75 left-truncatable primes in base 11, where largest is 2276005673
```

Find limit of recursion

Maximum recursion depth is memory dependent.

```
func recurse(n) {
   say n
   recurse(n+1)
}

recurse(0)
```

Output:

```
0
1
2
...
...
357077
357078
357079
```

Find palindromic numbers in both binary and ternary bases

```
var format = "%11s %24s %38s\n"
format.printf("decimal", "ternary", "binary")
format.printf(0, 0, 0)

for n in (0 .. 2e5) {
    var pal = n.base(3)||''
    var b3 = (pal + '1' + pal.flip)
    var b2 = Num(b3, 3).base(2)
    if (b2 == b2.flip) {
        format.printf(Num(b2, 2), b3, b2)
    }
}
```

Output:

```
    decimal                  ternary                                 binary
          0                        0                                      0
          1                        1                                      1
       6643                100010001                          1100111110011
    1422773             2200021200022                  10101101101010110110101
    5415589           101012010210101                 10100101010001010100101
90396755477  22122022220102222022122  10101000011000001000100000011000010101
```

Find prime n such that reversed n is also prime

```
say primes(500).grep { .reverse.is_prime }
```

Output:

```
[2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, 97, 101, 107, 113, 131, 149, 151, 157, 167, 179, 181, 191,
```

Find prime numbers of the form n*n*n+2

```
1..^200 -> map { _**3 + 2 }.grep {.is_prime}.say
```

Output:

```
[3, 29, 127, 24391, 91127, 250049, 274627, 328511, 357913, 571789, 1157627, 1442899, 1860869, 2146691,
```

Find square difference

```
var N = 1000

# Binary search
var n = bsearch_ge(1, N, {|k|
    k**2 - (k-1)**2 <=> N+1
})

# Closed-form
var m = ceil((N + 1 + N&1) / 2)

assert(n**2 - (n-1)**2 > N)
assert_eq(n, m)

say "#{n}^2 - #{n-1}^2 = #{n**2 - (n-1)**2}"
```

Output:

```
501^2 - 500^2 = 1001
```

Find squares n where n+1 is prime

```
1..1000.isqrt -> map { _**2 }.grep { is_prime(_+1) }.say
```

Output:

```
[1, 4, 16, 36, 100, 196, 256, 400, 576, 676]
```

Find the intersection of a line with a plane

```
struct Line {
    P0,       # point
    u⃗,        # ray
}

struct Plane {
    V0,       # point
    n⃗,        # normal
}

func dot_prod(a, b) { a »*« b -> sum }

func line_plane_intersection(L, P) {
    var cos = dot_prod(P.n⃗, L.u⃗) ->
     || return 'Vectors are orthogonal'
    var W = (L.P0 »-« P.V0)
    var SI = -(dot_prod(P.n⃗, W) / cos)
    W »+« (L.u⃗ »*» SI) »+« P.V0
}

say ('Intersection at point: ', line_plane_intersection(
         Line(P0: [0,0,10], u⃗: [0,-1,-1]),
        Plane(V0: [0,0, 5], n⃗: [0, 0, 1]),
))
```

Output:

```
Intersection at point: [0, -5, 5]
```

Find the intersection of two lines

```
func det(a, b, c, d) { a*d - b*c }

func intersection(ax, ay, bx, by,
                  cx, cy, dx, dy) {

    var detAB = det(ax,ay, bx,by)
    var detCD = det(cx,cy, dx,dy)

    var ΔxAB = (ax - bx)
    var ΔyAB = (ay - by)
    var ΔxCD = (cx - dx)
    var ΔyCD = (cy - dy)

    var x_numerator = det(detAB, ΔxAB, detCD, ΔxCD)
    var y_numerator = det(detAB, ΔyAB, detCD, ΔyCD)
    var denominator = det( ΔxAB, ΔyAB,  ΔxCD, ΔyCD)

    denominator == 0 && return 'lines are parallel'
    [x_numerator / denominator, y_numerator / denominator]
}

say ('Intersection point: ', intersection(4,0, 6,10, 0,3, 10,7))
say ('Intersection point: ', intersection(4,0, 6,10, 0,3, 10,7.1))
say ('Intersection point: ', intersection(0,0, 1,1, 1,2, 4,5))
```

Output:

```
Intersection point: [5, 5]
Intersection point: [2300/459, 2320/459]
Intersection point: lines are parallel
```

Find the last Sunday of each month

```
var dt = require('DateTime')
var (year=2016) = ARGV.map{.to_i}...

for i in (1 .. 12) {
    var date = dt.last_day_of_month(
        year  => year,
        month => i
    )

    while (date.dow != 7) {
        date = date.subtract(days => 1)
    }

    say date.ymd
}
```

Output:

```
$ sidef last_sunday.sf 2013
2013-01-27
2013-02-24
2013-03-31
2013-04-28
2013-05-26
2013-06-30
2013-07-28
2013-08-25
2013-09-29
2013-10-27
2013-11-24
2013-12-29
```

Find the missing permutation

```
func check_perm(arr) {
    var hash = Hash()
    hash.set_keys(arr...)
    arr.each { |s|
        {
            var t = (s.substr(1) + s.substr(0, 1))
            hash.has_key(t) || return t
        } * s.len
    }
}

var perms = %w(ABCD CABD ACDB DACB BCDA ACBD ADCB CDAB DABC BCAD CADB CDBA
               CBAD ABDC ADBC BDCA DCBA BACD BADC BDAC CBDA DBCA DCAB)

say check_perm(perms)
```

Output:

```
DBAC
```

Find words whose first and last three letters are equal

```
File("unixdict.txt").open_r.each {|word|
    word.len > 5 || next
    if (word.ends_with(word.first(3))) {
        say word
    }
}
```

Output:

```
antiperspirant
calendrical
einstein
hotshot
murmur
oshkosh
tartar
testes
```

First-class functions

```
func compose(f,g) {
    func (*args) {
        f(g(args...))
    }
}

var cube  = func(a) { a.pow(3) }
var croot = func(a) { a.root(3) }

var flist1 = [Num.method(:sin),  Num.method(:cos),  cube]
var flist2 = [Num.method(:asin), Num.method(:acos), croot]

for a,b (flist1 ~Z flist2) {
    say compose(a, b)(0.5)
}
```

Output:

```
0.5
0.5
0.5
```

First-class functions/Use numbers analogously

```
func multiplier(n1, n2) {
    func (n3) {
        n1 * n2 * n3
    }
}

var x  = 2.0
var xi = 0.5
var y  = 4.0
var yi = 0.25
var z  = (x + y)
var zi = (1 / (x + y))

var numbers = [x, y, z]
var inverses = [xi, yi, zi]

for f,g (numbers ~Z inverses) {
    say multiplier(f, g)(0.5)
}
```

Output:

```
0.5
0.5
0.5
```

First 9 prime Fibonacci number

```
say 12.by { .fib.is_prime }.map { .fib }
```

Output:

```
[2, 3, 5, 13, 89, 233, 1597, 28657, 514229, 433494437, 2971215073, 99194853094755497]
```

First class environments

```
func calculator({.is_one}   ) { 1 }
func calculator(n {.is_even}) { n / 2 }
func calculator(n           ) { 3*n + 1 }

func succ(this {_{:value}.is_one}, _) {
    return this
}

func succ(this, get_next) {
    this{:value} = get_next(this{:value})
    this{:count}++
    return this
}

var enviornments = (1..12 -> map {|i| Hash(value => i, count => 0) })

while (!enviornments.map{ _{:value} }.all { .is_one }) {
    say enviornments.map {|h| "%4s" % h{:value} }.join
    { |i|
        enviornments[i] = succ(enviornments[i], calculator)
    } << ^enviornments
}

say 'Counts'
say enviornments.map{ |h| "%4s" % h{:count} }.join
```

Output:

```
   1   2   3   4   5   6   7   8   9  10  11  12
   1   1  10   2  16   3  22   4  28   5  34   6
   1   1   5   1   8  10  11   2  14  16  17   3
   1   1  16   1   4   5  34   1   7   8  52  10
   1   1   8   1   2  16  17   1  22   4  26   5
   1   1   4   1   1   8  52   1  11   2  13  16
   1   1   2   1   1   4  26   1  34   1  40   8
   1   1   1   1   1   2  13   1  17   1  20   4
   1   1   1   1   1   1  40   1  52   1  10   2
   1   1   1   1   1   1  20   1  26   1   5   1
   1   1   1   1   1   1  10   1  13   1  16   1
   1   1   1   1   1   1   5   1  40   1   8   1
   1   1   1   1   1   1  16   1  20   1   4   1
   1   1   1   1   1   1   8   1  10   1   2   1
   1   1   1   1   1   1   4   1   5   1   1   1
   1   1   1   1   1   1   2   1  16   1   1   1
   1   1   1   1   1   1   1   1   8   1   1   1
   1   1   1   1   1   1   1   1   4   1   1   1
   1   1   1   1   1   1   1   1   2   1   1   1
Counts
   0   1   7   2   5   8  16   3  19   6  14   9
```

First missing positive

```
func missing(a) {
    a = a.grep { .is_pos }.sort.uniq
    1..Inf -> first { |n| a.shift != n }
}

for a in ([
    [1,2,0], [3,4,-1,1], [7,8,9,11,12],
    [3,2,1,99999999999999999999999],
    [1,1,2,1,1], [-5,-4,-3]
]) {
    say "First missing positive for #{a} = #{missing(a)}"
}
```

Output:

```
First missing positive for [1, 2, 0] = 3
First missing positive for [3, 4, -1, 1] = 2
First missing positive for [7, 8, 9, 11, 12] = 1
First missing positive for [3, 2, 1, 99999999999999999999999] = 4
First missing positive for [1, 1, 2, 1, 1] = 3
First missing positive for [-5, -4, -3] = 1
```

First perfect square in base n with n unique digits

```
func first_square(b) {

    var start = [1, 0, (2..^b)...].flip.map_kv{|k,v| v * b**k }.sum.isqrt

    start..Inf -> first_by {|k|
        k.sqr.digits(b).freq.len == b
    }.sqr
}

for b in (2..16) {
    var s = first_square(b)
    printf("Base %2d: %10s² == %s\n", b, s.isqrt.base(b), s.base(b))
}
```

Output:

```
Base  2:         10² == 100
Base  3:         22² == 2101
Base  4:         33² == 3201
Base  5:        243² == 132304
Base  6:        523² == 452013
Base  7:       1431² == 2450361
Base  8:       3344² == 13675420
Base  9:      11642² == 136802574
Base 10:      32043² == 1026753849
Base 11:     111453² == 1240a536789
Base 12:     3966b9² == 124a7b538609
Base 13:    3828943² == 10254773ca86b9
Base 14:    3a9db7c² == 10269b8c57d3a4
Base 15:   1012b857² == 102597bace836d4
Base 16:   404a9d9b² == 1025648cfea37bd9
```

First power of 2 that has leading decimal digits of 12

```
func farey_approximations(r, callback) {

    var (a1 = r.int, b1 = 1)
    var (a2 = a1+1,  b2 = 1)

    loop {
        var a3 = a1+a2
        var b3 = b1+b2

        if (a3 < r*b3) {
            (a1, b1) = (a3, b3)
        }
        else {
            (a2, b2) = (a3, b3)
        }

        callback(a3 / b3)
    }
}

func p(L, nth) {

    define ln2  = log(2)
    define ln5  = log(5)
    define ln10 = log(10)

    var t = L.len-1

    func isok(n) {
        floor(exp(ln2*(n - floor((n*ln2)/ln10) + t) + ln5*(t - floor((n*ln2)/ln10)))) == L
    }

    var deltas = gather {
        farey_approximations(ln2/ln10, {|r|
            take(r.de) if (r.de.len == L.len)
            break      if (r.de.len >  L.len)
        })
    }.sort.uniq

    var c = 0
    var k = (1..Inf -> first(isok))

    loop {
        return k if (++c == nth)
        k += (deltas.first {|d| isok(k+d) } \\ die "error: #{k}")
    }
}

var tests = [
    [12, 1],
    [12, 2],
    [123, 45],
    [123, 12345],
    [123, 678910],

    # extra
    [1234, 10000],
    [12345, 10000],
]

for a,b in (tests) {
    say "p(#{a}, #{b}) = #{p(a,b)}"
}
```

Output:

```
p(12, 1) = 7
p(12, 2) = 80
p(123, 45) = 12710
p(123, 12345) = 3510491
p(123, 678910) = 193060223
p(1234, 10000) = 28417587
p(12345, 10000) = 284166722
```

Five weekends

```
require('DateTime')

var happymonths = []
var workhardyears = []
var longmonths = [1, 3, 5, 7, 8, 10, 12]

for year in (1900 .. 2100) {
   var countmonths = 0
   longmonths.each { |month|
        var dt = %O<DateTime>.new(
            year => year,
            month => month,
            day   => 1
        )

        if (dt.day_of_week == 5) {
            countmonths++
            var yearfound = dt.year
            var monthfound = dt.month_name
            happymonths.append(join("  ", yearfound, monthfound))
      }
   }

    if (countmonths == 0) {
        workhardyears.append(year)
    }
}

say "There are #{happymonths.len} months with 5 full weekends!"
say "The first 5 and the last 5 of them are:"
say happymonths.first(5).join("\n")
say happymonths.last(5).join("\n")
say "No long weekends in the following #{workhardyears.len} years:"
say workhardyears.join(",")
```

Output:

```
There are 201 months with 5 full weekends!
The first 5 and the last 5 of them are:
1901  March
1902  August
1903  May
1904  January
1904  July
2097  March
2098  August
2099  May
2100  January
2100  October
No long weekends in the following 29 years:
1900,1906,1917,1923,1928,1934,1945,1951,1956,1962,1973,1979,1984,1990,2001,2007,2012,2018,2029,2035,204
```

Fivenum

```
func fourths(e) {
    var t = ((e>>1) / 2)
    [0, t, e/2, e - t, e]
}

func fivenum(nums) {
    var x = nums.sort
    var d = fourths(x.end)

    ([x[d.map{.floor}]] ~Z+ [x[d.map{.ceil}]]) »/» 2
}

var nums = [
    [15, 6, 42, 41, 7, 36, 49, 40, 39, 47, 43],
    [36, 40, 7, 39, 41, 15], [
    0.14082834,  0.09748790,  1.73131507,  0.87636009, -1.95059594,
    0.73438555, -0.03035726,  1.46675970, -0.74621349, -0.72588772,
    0.63905160,  0.61501527, -0.98983780, -1.00447874, -0.62759469,
    0.66206163,  1.04312009, -0.10305385,  0.75775634,  0.32566578,
]]

nums.each { say fivenum(_).join(', ') }
```

Output:

```
6, 25.5, 40, 42.5, 49
7, 15, 37.5, 40, 41
-1.95059594, -0.676741205, 0.23324706, 0.746070945, 1.73131507
```

FizzBuzz

Structured:

```
{ |i|
    if (i %% 3) {
        print "Fizz"
        i %% 5 && print "Buzz"
        print "\n"
    }
    elsif (i %% 5) { say "Buzz" }
    else  { say i }
} << 1..100
```

Declarative:

```
func fizzbuzz({ _ %% 15 }) { "FizzBuzz" }
func fizzbuzz({ _ %%  5 }) {     "Buzz" }
func fizzbuzz({ _ %%  3 }) {     "Fizz" }
func fizzbuzz(        n  ) {          n }

for n in (1..100) { say fizzbuzz(n) }
```

One-liner:

```
{|i|say "#{<Fizz>[i%3]}#{<Buzz>[i%5]}"||i}*100;
```

Flatten a list

```
func flatten(a) {
    var flat = []
    a.each { |item|
        flat += (item.kind_of(Array) ? flatten(item) : [item])
    }
    return flat
}

var arr = [[1], 2, [[3,4], 5], [[[]]], [[[6]]], 7, 8, []]
say flatten(arr)      # used-defined function
say arr.flatten       # built-in Array method
```

Flow-control structures

goto

```
say "Hello"
goto :world
say "Never printed"
@:world
say "World"
```

Output:

```
Hello
World
```

Floyd's triangle

```
func floyd(rows, n=1) {
    var max = Math.range_sum(1, rows)
    var widths = (max-rows .. max-1 -> map{.+n->to_s.len})
    { |r|
        say %'#{1..r -> map{|i| "%#{widths[i-1]}d" % n++}.join(" ")}'
    } << 1..rows
}

floyd(5)     # or: floyd(5, 88)
floyd(14)    # or: floyd(14, 900)
```

Output:

```
 1
 2  3
 4  5  6
 7  8  9 10
11 12 13 14 15
 1
 2  3
 4  5  6
 7  8  9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44  45
46 47 48 49 50 51 52 53  54  55
56 57 58 59 60 61 62 63  64  65  66
67 68 69 70 71 72 73 74  75  76  77  78
79 80 81 82 83 84 85 86  87  88  89  90  91
92 93 94 95 96 97 98 99 100 101 102 103 104 105
```

Floyd-Warshall algorithm

```
func floyd_warshall(n, edge) {
    var dist = n.of {|i| n.of { |j| i == j ? 0 : Inf }}
    var nxt  = n.of { n.of(nil) }
    for u,v,w in edge {
        dist[u-1][v-1] = w
         nxt[u-1][v-1] = v-1
    }

    [^n] * 3 -> cartesian { |k, i, j|
        if (dist[i][j] > dist[i][k]+dist[k][j]) {
            dist[i][j] = dist[i][k]+dist[k][j]
            nxt[i][j] = nxt[i][k]
        }
    }

    var summary = "pair     dist    path\n"
    for i,j (^n ~X ^n) {
        i==j && next
        var u = i
        var path = [u]
        while (u != j) {
            path << (u = nxt[u][j])
        }
        path.map!{|u| u+1 }.join!(" -> ")
        summary += ("%d -> %d  %4d     %s\n" % (i+1, j+1, dist[i][j], path))
    }

    return summary
}

var n = 4
var edge = [[1, 3, -2], [2, 1, 4], [2, 3, 3], [3, 4, 2], [4, 2, -1]]
print floyd_warshall(n, edge)
```

Output:

```
pair     dist    path
1 -> 2    -1     1 -> 3 -> 4 -> 2
1 -> 3    -2     1 -> 3
1 -> 4     0     1 -> 3 -> 4
2 -> 1     4     2 -> 1
2 -> 3     2     2 -> 1 -> 3
2 -> 4     4     2 -> 1 -> 3 -> 4
3 -> 1     5     3 -> 4 -> 2 -> 1
3 -> 2     1     3 -> 4 -> 2
3 -> 4     2     3 -> 4
4 -> 1     3     4 -> 2 -> 1
4 -> 2    -1     4 -> 2
4 -> 3     1     4 -> 2 -> 1 -> 3
```

Forest fire

```
define w = `tput cols`.to_i
define h = `tput lines`.to_i
define r = "\033[H"

define red = "\033[31m"
define green = "\033[32m"
define yellow = "\033[33m"

define chars = [' ', green+'*', yellow+'&', red+'&']

define tree_prob = 0.05
define burn_prob = 0.0002

enum |Empty, Tree, Heating, Burning|

define dirs = [
    %n(-1 -1), %n(-1 0), %n(-1 1), %n(0 -1),
    %n(0   1), %n(1 -1), %n(1  0), %n(1  1),
]

var forest = h.of { w.of { 1.rand < tree_prob ? Tree : Empty } }

var range_h = h.range
var range_w = w.range

func iterate {
    var new = h.of{ w.of(0) }
    for i in range_h {
        for j in range_w {
            given (new[i][j] = forest[i][j]) {
              when (Tree) {
                1.rand < burn_prob && (new[i][j] = Heating; next)
                dirs.each { |pair|
                    var y = pair[0]+i
                    range_h.contains(y) || next
                    var x = pair[1]+j
                    range_w.contains(x) || next
                    forest[y][x] == Heating && (new[i][j] = Heating; break)
                }
              }
              when (Heating)            { new[i][j] = Burning }
              when (Burning)            { new[i][j] = Empty   }
              case (1.rand < tree_prob) { new[i][j] = Tree    }
            }
        }
    }
    forest = new
}

STDOUT.autoflush(true)

func init_forest {
    print r
    forest.each { |row|
        print chars[row]
        print "\033[E\033[1G"
    }
    iterate()
}

loop { init_forest() }
```

OO version:

```
define RED = "\e[1;31m"
define YELLOW = "\e[1;33m"
define GREEN = "\e[1;32m"

define DIRS = [
    [-1, -1], [0, -1], [1, -1],
    [-1,  0],          [1,  0],
    [-1,  1], [0,  1], [1,  1],
]

enum (Empty, Tree, Heating, Burning)
define pix = [' ', GREEN + "*", YELLOW + "*", RED + "*"]

class Forest(p=0.01, f=0.001, height, width) {

    has coords = []
    has spot = []
    has neighbors = []

    method init {
        coords = (^height ~X ^width)
        spot = height.of { width.of { [true, false].pick ? Tree : Empty } }
        self.init_neighbors
    }

    method init_neighbors {
        for i,j in coords {
            neighbors[i][j] = gather {
                for dir in DIRS {
                    take(\(spot[i + dir[0]][j + dir[1]] \\ next))
                 }
            }
        }
    }

    method step {
        var heat = []

        for i,j in coords {
            given (spot[i][j]) {
                when Empty   { spot[i][j] = Tree    if (1.rand < p) }
                when Tree    { spot[i][j] = Heating if (1.rand < f) }
                when Heating { spot[i][j] = Burning; heat << [i, j] }
                when Burning { spot[i][j] = Empty }
            }
        }

        for i,j in heat {
            neighbors[i][j].each { |ref|
                *ref = Heating if (*ref == Tree)
            }
        }
    }

    method show {
        { |i|
            say pix[spot[i]]
        } << ^height
    }
}

STDOUT.autoflush(true)
var(height, width) = `stty size`.nums.map{.dec}...

var forest = Forest(height: height, width: width)
```

```
print "\e[2J"

loop {
    print "\e[H"
    forest.show
    forest.step
}
```

Fork

```
var x = 42
{ x += 1; say x }.fork.wait      # x is 43 here
say x                            # but here is still 42
```

Formatted numeric output

```
printf("%09.3f\n", 7.125)
```

or

```
say ("%09.3f" % 7.125)
```

Output:

```
00007.125
```

Fortunate numbers

```
func fortunate(n) {
    var P = n.pn_primorial
    2..Inf -> first {|m| P+m -> is_prob_prime }
}

var limit = 50
var uniq = Set()
var all = []

for (var n = 1; uniq.len < 2*limit; ++n) {
    var m = fortunate(n)
    all << m
    uniq << m
}

say "Fortunate numbers for n = 1..#{limit}:"
say all.first(limit)

say "\n#{limit} Fortunate numbers, sorted with duplicates removed:"
say uniq.sort.first(limit)
```

Output:

```
Fortunate numbers for n = 1..50:
[3, 5, 7, 13, 23, 17, 19, 23, 37, 61, 67, 61, 71, 47, 107, 59, 61, 109, 89, 103, 79, 151, 197, 101, 103

50 Fortunate numbers, sorted with duplicates removed:
[3, 5, 7, 13, 17, 19, 23, 37, 47, 59, 61, 67, 71, 79, 89, 101, 103, 107, 109, 127, 151, 157, 163, 167,
```

Forward difference

```
func dif(arr) {
    gather {
        for i (0 ..^ arr.end) {
            take(arr[i+1] - arr[i])
        }
    }
}

func difn(n, arr) {
    { arr = dif(arr) } * n
    arr
}

say dif([1, 23, 45, 678])       # => [22, 22, 633]
say difn(2, [1, 23, 45, 678])   # => [0, 611]
```

Four bit adder

```
func bxor(a, b) {
  (~a & b) | (a & ~b)
}

func half_adder(a, b) {
  return (bxor(a, b), a & b)
}

func full_adder(a, b, c) {
  var (s1, c1) = half_adder(a, c)
  var (s2, c2) = half_adder(s1, b)
  return (s2, c1 | c2)
}

func four_bit_adder(a, b) {
  var (s0, c0) = full_adder(a[0], b[0], 0)
  var (s1, c1) = full_adder(a[1], b[1], c0)
  var (s2, c2) = full_adder(a[2], b[2], c1)
  var (s3, c3) = full_adder(a[3], b[3], c2)
  return ([s3,s2,s1,s0].join, c3.to_s)
}

say " A    B      A      B   C    S  sum"
for a in ^16 {
  for b in ^16 {
    var(abin, bbin) = [a,b].map{|n| "%04b"%n->chars.flip.map{.to_i} }...
    var(s, c) = four_bit_adder(abin, bbin)
    printf("%2d + %2d = %s + %s = %s %s = %2d\n",
        a, b, abin.join, bbin.join, c, s, "#{c}#{s}".bin)
    }
}
```

Output:

```
 A    B      A      B   C    S  sum
 0 +  0 = 0000 + 0000 = 0 0000 =  0
 0 +  1 = 0000 + 0001 = 0 0001 =  1
 0 +  2 = 0000 + 0010 = 0 0010 =  2
 0 +  3 = 0000 + 0011 = 0 0011 =  3
 0 +  4 = 0000 + 0100 = 0 0100 =  4
...
 7 + 13 = 0111 + 1101 = 1 0100 = 20
 7 + 14 = 0111 + 1110 = 1 0101 = 21
 7 + 15 = 0111 + 1111 = 1 0110 = 22
 8 +  0 = 1000 + 0000 = 0 1000 =  8
 8 +  1 = 1000 + 0001 = 0 1001 =  9
 8 +  2 = 1000 + 0010 = 0 1010 = 10
...
15 + 12 = 1111 + 1100 = 1 1011 = 27
15 + 13 = 1111 + 1101 = 1 1100 = 28
15 + 14 = 1111 + 1110 = 1 1101 = 29
15 + 15 = 1111 + 1111 = 1 1110 = 30
```

Four is magic

```
func cardinal(n) {
    static lingua_en = frequire("Lingua::EN::Numbers")
    lingua_en.num2en(n) - / and|,/g
}

func four_is_magic(n) {
    var str = ""
    loop {
       str += (cardinal(n) + " is ")
       if (n == 4) {
           str += "magic."
           break
       } else {
           n = cardinal(n).len
           str += (cardinal(n) + ", ")
       }
   }
   str.tc
}

[0, 4, 6, 11, 13, 75, 337, -164, 9_876_543_209].each { |n|
    say four_is_magic(n)
}
```

Output:

```
Zero is four, four is magic.
Four is magic.
Six is three, three is five, five is four, four is magic.
Eleven is six, six is three, three is five, five is four, four is magic.
Thirteen is eight, eight is five, five is four, four is magic.
Seventy-five is twelve, twelve is six, six is three, three is five, five is four, four is magic.
Three hundred thirty-seven is twenty-six, twenty-six is ten, ten is three, three is five, five is four,
Negative one hundred sixty-four is thirty-one, thirty-one is ten, ten is three, three is five, five is
Nine billion eight hundred seventy-six million five hundred forty-three thousand two hundred nine is ni
```

Four sides of square

```
var n = 5

[n.of(1), (n-2).of([1, (n-2).of(0)..., 1])..., n.of(1)].each {|row|
    say row.join(' ')
}
```

Output:

```
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

Fractal tree

```
func tree(img, x, y, scale=6/10, len=400, angle=270) {

    len < 1 && return()

    img.moveTo(x, y)
    img.angle(angle)
    img.line(len)

    var (x1, y1) = img.curPos
    tree(img, x1, y1, scale, len*scale, angle+35)
    tree(img, x1, y1, scale, len*scale, angle-35)
}

require('GD::Simple')

var (width=1000, height=1000)
var img = %s<GD::Simple>.new(width, height)
img.fgcolor('black')
img.penSize(1, 1)

tree(img, width/2, height)

File('tree.png').write(img.png, :raw)
```

Fractran

```
var str ="17/91, 78/85, 19/51, 23/38, 29/33, 77/29, 95/23, 77/19, 1/17, 11/13, 13/11, 15/14, 15/2,
55/1"
const FractalProgram = str.split(',').map{.num}      #=> array of rationals

func runner(n, callback) {
    var num = 2
    n.times {
        callback(num *= FractalProgram.find { |f| f * num -> is_int })
    }
}

func prime_generator(n, callback) {
    var x = 0;
    runner(Inf, { |num|
        var l = num.log2
        if (l.floor == l) {
            callback(l.int)
            ++x == n && return nil
        }
    })
}

STDOUT.autoflush(true)

runner(20, {|n| print (n, ' ') })
print "\n"

prime_generator(20, {|n| print (n, ' ') })
print "\n"
```

Output:

```
15 825 725 1925 2275 425 390 330 290 770 910 170 156 132 116 308 364 68 4 30
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
```

Free polyominoes enumeration

```
func translate2origin(poly) {
  # Finds the min x and y coordiate of a Polyomino.
  var minx = poly.map(:head).min
  var miny = poly.map(:tail).min
  poly.map {|p| [p.head-minx, p.tail-miny] }.sort
}

func rotate90(x,y) { [y, -x] }
func reflect(x,y)  { [-x, y] }

# All the plane symmetries of a rectangular region.
func rotations_and_reflections(poly) {
    gather {
        take(poly)
        take(poly.map!{ rotate90(_...) })
        take(poly.map!{ rotate90(_...) })
        take(poly.map!{ rotate90(_...) })
        take(poly.map!{  reflect(_...) })
        take(poly.map!{ rotate90(_...) })
        take(poly.map!{ rotate90(_...) })
        take(poly.map!{ rotate90(_...) })
    }
}

func canonical(poly) {
  rotations_and_reflections(poly).map{|pl| translate2origin(pl) }
}

# All four points in Von Neumann neighborhood.
func contiguous(x, y) {
  [[x-1, y], [x+1, y], [x, y-1], [x, y+1]]
}

# Finds all distinct points that can be added to a Polyomino.
func new_points(poly) {
  var points = Set()
  poly.each { points << contiguous(_...)... }
  points - poly
}

func new_polys(polys) {
  var pattern = Set()
  polys.map { |poly|
    gather {
      new_points(poly).each { |point|
        var pl = translate2origin(poly + [point])
        next if pattern.has(pl)
        take canonical(pl).each{ pattern << _ }.min
      }
    }...
  }
}

# Generates polyominoes of rank n recursively.
func rank(n) {
  given (n) {
```

```
    when (0) { [[]] }
    when (1) { [[[0,0]]] }
    else     { new_polys(rank(n-1)) }
  }
}

# Generates a textual representation of a Polyomino.
func text_representation(poly) {
  var table = Hash()
  for x,y in (poly) { table{[x,y]} = '#' }
  var maxx = poly.map(:head).max
  var maxy = poly.map(:tail).max
  (0..maxx).map{|x| (0..maxy).map{|y| table{[x,y]} \\ ' ' }.join }
}

say 8.of { rank(_).len }

var n = (ARGV[0] ? ARGV[0].to_i : 5)
say ("\nAll free polyominoes of rank %d:" % n)
rank(n).sort.each{|poly| say text_representation(poly).join("\n")+"\n" }
```

French Republican calendar

```
require('DateTime')

var month_names = %w(
  Vendémiaire Brumaire Frimaire  Nivôse   Pluviôse  Ventôse
  Germinal    Floréal  Prairial  Messidor Thermidor Fructidor
)

var intercalary = [
                   'Fête de la vertu',
                   'Fête du génie',
                   'Fête du travail',
                   "Fête de l'opinion",
                   'Fête des récompenses',
                   'Fête de la Révolution',
                  ]

var i_cal_month = 13
var epoch = %O<DateTime>.new(year => 1792, month => 9, day => 22)

var month_nums = Hash(month_names.kv.map {|p| [p[1], p[0]+1] }.flat...)
var i_cal_nums = Hash(intercalary.kv.map {|p| [p[1], p[0]+1] }.flat...)

func is_republican_leap_year(Number year) -> Bool {
    var y = (year + 1)
    !!(4.divides(y) && (!100.divides(y) || 400.divides(y)))
}

func Republican_to_Gregorian(String rep_date) -> String {
    static months   = month_names.map { .escape }.join('|')
    static intercal = intercalary.map { .escape }.join('|')
    static re       = Regex("^
        \\s* (?:
                (?<ic> #{intercal})
              | (?<day> \\d+) \\s+ (?<month> #{months})
            )
        \\s+ (?<year> \\d+)
        \\s*
      \\z", 'x')
```

```
    var m = (rep_date =~ re)
    m || die "Republican date not recognized: '#{rep_date}'"

    var ncap = m.named_captures

    var day1   = Number(ncap{:ic}    ? i_cal_nums{ncap{:ic}}    : ncap{:day})
    var month1 = Number(ncap{:month} ? month_nums{ncap{:month}} : i_cal_month)
    var year1  = Number(ncap{:year})

    var days_since_epoch = (365*(year1 - 1) + 30*(month1 - 1) + (day1 - 1))

    var leap_days = (1 ..^ year1 -> grep { is_republican_leap_year(_) })
    epoch.clone.add(days => (days_since_epoch + leap_days)).strftime("%Y-%m-%d")
}

func Gregorian_to_Republican(String greg_date) -> String {
    var m = (greg_date =~ /^(\d{4})-(\d{2})-(\d{2})\z/)
    m || die "Gregorian date not recognized: '#{greg_date}'"

    var g = %O<DateTime>.new(year => m[0], month => m[1], day => m[2])
    var days_since_epoch = epoch.delta_days(g).in_units('days')
    days_since_epoch < 0 && die "unexpected error"
    var (year, days) = (1, days_since_epoch)

    loop {
        var year_length = (365 + (is_republican_leap_year(year) ? 1 : 0))
        days < year_length && break
        days -= year_length
        year += 1;
    }

    var day0   = (days % 30)
    var month0 = (days - day0)/30

    var (day1, month1) = (day0 + 1, month0 + 1)

    (month1 == i_cal_month
        ? "#{intercalary[day0]} #{year}"
        : "#{day1} #{month_names[month0]} #{year}")
}

for line in DATA {

    line.sub!(/\s*\#.+\R?\z/, '')

    var m = (line =~ /^(\d{4})-(\d{2})-(\d{2})\s+(\S.+?\S)\s*$/)
    m || die "error for: #{line.dump}"

    var g = "#{m[0]}-#{m[1]}-#{m[2]}"
    var r = m[3]

    var r2g = Republican_to_Gregorian(r)
    var g2r = Gregorian_to_Republican(g)

    #say "#{r} -> #{r2g}"
    #say "#{g} -> #{g2r}"

    if ((g2r != r) || (r2g != g)) {
        die "1-way error"
    }

    if ((Gregorian_to_Republican(r2g) != r) ||
        (Republican_to_Gregorian(g2r) != g)
    ) {
        die "2-way error"
```

```
    }
}

say 'All tests successful.'

__DATA__
1792-09-22  1 Vendémiaire 1
1795-05-20  1 Prairial 3
1799-07-15  27 Messidor 7
1803-09-23  Fête de la Révolution 11
1805-12-31  10 Nivôse 14
1871-03-18  27 Ventôse 79
1944-08-25  7 Fructidor 152
2016-09-19  Fête du travail 224
1871-05-06  16 Floréal 79   # Paris Commune begins
1871-05-23  3 Prairial 79   # Paris Commune ends
1799-11-09  18 Brumaire 8   # Revolution ends by Napoléon coup
1804-12-02  11 Frimaire 13  # Republic   ends by Napoléon coronation
1794-10-30  9 Brumaire 3    # École Normale Supérieure established
1794-07-27  9 Thermidor 2   # Robespierre falls
1799-05-27  8 Prairial 7    # Fromental Halévy born
1792-09-22  1 Vendémiaire 1
1793-09-22  1 Vendémiaire 2
1794-09-22  1 Vendémiaire 3
1795-09-23  1 Vendémiaire 4
1796-09-22  1 Vendémiaire 5
1797-09-22  1 Vendémiaire 6
1798-09-22  1 Vendémiaire 7
1799-09-23  1 Vendémiaire 8
1800-09-23  1 Vendémiaire 9
1801-09-23  1 Vendémiaire 10
1802-09-23  1 Vendémiaire 11
1803-09-24  1 Vendémiaire 12
1804-09-23  1 Vendémiaire 13
1805-09-23  1 Vendémiaire 14
1806-09-23  1 Vendémiaire 15
1807-09-24  1 Vendémiaire 16
1808-09-23  1 Vendémiaire 17
1809-09-23  1 Vendémiaire 18
1810-09-23  1 Vendémiaire 19
1811-09-24  1 Vendémiaire 20
2015-09-23  1 Vendémiaire 224
2016-09-22  1 Vendémiaire 225
2017-09-22  1 Vendémiaire 226
```

Output:

```
All tests successful.
```

Frobenius numbers

```
func frobenius_number(n) {
    prime(n) * prime(n+1) - prime(n) - prime(n+1)
}

say gather {
    1..Inf -> each {|k|
        var n = frobenius_number(k)
        break if (n >= 10_000)
        take(n)
    }
}
```

Output:

```
[1, 7, 23, 59, 119, 191, 287, 395, 615, 839, 1079, 1439, 1679, 1931, 2391, 3015, 3479, 3959, 4619, 5039
```

FTP

```
require('Net::FTP')

var ftp = %O<Net::FTP>.new('ftp.ed.ac.uk', Passive => 1)
ftp.login('anonymous','email@example.com')
ftp.cwd('pub/courses')
[ftp.dir].each {|line| say line }
ftp.binary;   # set binary mode
ftp.get("make.notes.tar")
ftp.quit
```

Function composition

```
func compose(f, g) {
    func(x) { f(g(x)) }
}

var fg = compose(func(x){ sin(x) }, func(x){ cos(x) })
say fg(0.5)      # => 0.769196354841008421852514758051 07
```

Function definition

```
func multiply(a, b) {
    a * b
}
```

Function frequency

Sidef provides full access to its parser, allowing us to inspect all the declarations within a program.

```
func foo { }
func bar { }

foo(); foo(); foo()
bar(); bar();

var data = Perl.to_sidef(Parser{:vars}{:main}).flatten

data.sort_by { |v| -v{:count} }.first(10).each { |entry|
    if (entry{:type} == :func) {
        say ("Function `#{entry{:name}}` (declared at line",
             " #{entry{:line}}) is used #{entry{:count}} times")
    }
}
```

Output:

```
Function `foo` (declared at line 1) is used 3 times
Function `bar` (declared at line 2) is used 2 times
```

Fusc sequence

```
func fusc(n) is cached {

    return 0 if n.is_zero
    return 1 if n.is_one

    n.is_even ? fusc(n/2) : (fusc((n-1)/2) + fusc(((n-1)/2)+1))
}

say ("First 61 terms of the Stern-Brocot sequence: ", 61.of(fusc).join(' '))

say "\nIndex and value for first term longer than any previous:"
printf("%15s : %s\n", "Index", "Value");

var (index=0, len=0)

5.times {
    index = (index..Inf -> first_by { fusc(_).len > len })
    len = fusc(index).len
    printf("%15s : %s\n", index.commify, fusc(index).commify)
}
```

Output:

```
First 61 terms of the Stern-Brocot sequence: 0 1 1 2 1 3 2 3 1 4 3 5 2 5 3 4 1 5 4 7 3 8 5 7 2 7 5 8 3

Index and value for first term longer than any previous:
          Index : Value
              0 : 0
             37 : 11
          1,173 : 108
         35,499 : 1,076
        699,051 : 10,946
```

Gamma function

```
var a = [ 1.00000_00000_00000_00000,  0.57721_56649_01532_86061, -0.65587_80715_20253_88108,
          -0.04200_26350_34095_23553,  0.16653_86113_82291_48950, -0.04219_77345_55544_33675,
          -0.00962_19715_27876_97356,  0.00721_89432_46663_09954, -0.00116_51675_91859_06511,
          -0.00021_52416_74114_95097,  0.00012_80502_82388_11619, -0.00002_01348_54780_78824,
          -0.00000_12504_93482_14267,  0.00000_11330_27231_98170, -0.00000_02056_33841_69776,
           0.00000_00061_16095_10448,  0.00000_00050_02007_64447, -0.00000_00011_81274_57049,
           0.00000_00001_04342_67117,  0.00000_00000_07782_26344, -0.00000_00000_03696_80562,
           0.00000_00000_00510_03703, -0.00000_00000_00020_58326, -0.00000_00000_00005_34812,
           0.00000_00000_00001_22678, -0.00000_00000_00000_11813,  0.00000_00000_00000_00119,
           0.00000_00000_00000_00141, -0.00000_00000_00000_00023,  0.00000_00000_00000_00002 ]

func gamma(x) {
    var y = (x - 1)
    1 / a.reverse.reduce {|sum, an| sum*y + an}
}

for i in 1..10 {
    say ("%.14e" % gamma(i/3))
}
```

Output:

```
2.67893853470775e+00
1.35411793942640e+00
1.00000000000000e+00
8.92979511569249e-01
9.02745292950934e-01
1.00000000000000e+00
1.19063934875900e+00
1.50457548825154e+00
1.99999999999397e+00
2.77815847933858e+00
```

Lanczos approximation:

```
func gamma(z) {
    var epsilon = 0.0000001
    func withinepsilon(x) {
        abs(x - abs(x)) <= epsilon
    }

    var p = [
        676.5203681218851,     -1259.1392167224028,
        771.32342877765313,    -176.61502916214059,
        12.507343278686905,    -0.13857109526572012,
        9.9843695780195716e-6,  1.5056327351493116e-7,
    ]

    var result = 0
    const pi = Num.pi

    if (z.re < 0.5) {
        result = (pi / (sin(pi * z) * gamma(1 - z)))
    }
    else {
        z -= 1
        var x = 0.99999999999980993

        p.each_kv { |i, v|
            x += v/(z + i + 1)
        }

        var t = (z + p.len - 0.5)
        result = (sqrt(pi*2) * t**(z+0.5) * exp(-t) * x)
    }

    withinepsilon(result.im) ? result.re : result
}

for i in 1..10 {
    say ("%.14e" % gamma(i/3))
}
```

Output:

```
2.67893853470774e+00
1.35411793942640e+00
1.00000000000000e+00
8.92979511569252e-01
9.02745292950931e-01
1.00000000000000e+00
1.19063934875900e+00
1.50457548825155e+00
2.00000000000000e+00
2.77815848043767e+00
```

A simpler implementation:

```
define ℯ = Num.e
define τ = Num.tau

func Γ(t) {
    t < 20 ? (__FUNC__(t + 1) / t)
           : (sqrt(τ*t) * pow(t/ℯ + 1/(12*ℯ*t), t) / t)
}

for i in (1..10) {
    say ("%.14e" % Γ(i/3))
}
```

Output:

```
2.67893831294932e+00
1.35411783267848e+00
9.99999913007168e-01
8.92979437649773e-01
9.02745221785653e-01
9.99999913007168e-01
1.19063925019970e+00
1.50457536964275e+00
1.99999982601434e+00
2.77815825046596e+00
```

Gapful numbers

Concept extended to other bases:

```
func is_gapful(n, base=10) {
    n.is_div(base*floor(n / base**n.ilog(base)) + n%base)
}

var task = [
    "(Required) The first %s gapful numbers (>= %s)", 30, 1e2, 10,
    "(Required) The first %s gapful numbers (>= %s)", 15, 1e6, 10,
    "(Required) The first %s gapful numbers (>= %s)", 10, 1e9, 10,
    "(Extra) The first %s gapful numbers (>= %s)", 10, 987654321, 10,
    "(Extra) The first %s gapful numbers (>= %s)", 10, 987654321, 12,
]

task.each_slice(4, {|title, n, from, b|
    say sprintf("\n#{title} for base #{b}:", n, from.commify)
    say (from..Inf -> lazy.grep{ is_gapful(_,b) }.first(n).join(' '))
})
```

Output:

```
(Required) The first 30 gapful numbers (>= 100) for base 10:
100 105 108 110 120 121 130 132 135 140 143 150 154 160 165 170 176 180 187 190 192 195 198 200 220 225

(Required) The first 15 gapful numbers (>= 1,000,000) for base 10:
1000000 1000005 1000008 1000010 1000016 1000020 1000021 1000030 1000032 1000034 1000035 1000040 1000050

(Required) The first 10 gapful numbers (>= 1,000,000,000) for base 10:
1000000000 1000000001 1000000005 1000000008 1000000010 1000000016 1000000020 1000000027 1000000030 1000

(Extra) The first 10 gapful numbers (>= 987,654,321) for base 10:
987654330 987654334 987654336 987654388 987654420 987654485 987654510 987654513 987654592 987654600

(Extra) The first 10 gapful numbers (>= 987,654,321) for base 12:
987654325 987654330 987654336 987654360 987654368 987654384 987654388 987654390 987654393 987654395
```

Gauss-Jordan matrix inversion

Uses the `rref(A)` function from Reduced row echelon form.

```
func gauss_jordan_invert (M) {

    var I = M.len.of {|i|
        M.len.of {|j|
            i == j ? 1 : 0
        }
    }

    var A = gather {
        ^M -> each {|i| take(M[i] + I[i]) }
    }

    rref(A).map { .last(M.len) }
}

var A = [
    [-1, -2, 3, 2],
    [-4, -1, 6, 2],
    [ 7, -8, 9, 1],
    [ 1, -2, 1, 3],
]

say gauss_jordan_invert(A).map {
    .map { "%6s" % .as_rat }.join("  ")
}.join("\n")
```

Output:

```
-21/23   17/69  13/138   19/46
-38/23   15/23    1/23   15/23
-16/23   25/69  11/138    9/46
-13/23   16/69   -2/69   13/23
```

Gaussian elimination

Uses the `rref(A)` function from Reduced row echelon form.

```
func gauss_jordan_solve (a, b) {

    var A = gather {
        ^b -> each {|i| take(a[i] + b[i]) }
    }

    rref(A).map{ .last }
}

var a = [
    [ 1.00, 0.00, 0.00,  0.00,  0.00,   0.00 ],
    [ 1.00, 0.63, 0.39,  0.25,  0.16,   0.10 ],
    [ 1.00, 1.26, 1.58,  1.98,  2.49,   3.13 ],
    [ 1.00, 1.88, 3.55,  6.70, 12.62,  23.80 ],
    [ 1.00, 2.51, 6.32, 15.88, 39.90, 100.28 ],
    [ 1.00, 3.14, 9.87, 31.01, 97.41, 306.02 ],
]

var b = [ -0.01, 0.61, 0.91, 0.99, 0.60, 0.02 ]

var G = gauss_jordan_solve(a, b)
say G.map { "%27s" % .as_rat }.join("\n")
```

Output:

```
                   -1/100
  655870882787/409205648497
 -660131804286/409205648497
  509663229635/409205648497
 -200915766608/409205648497
   26909648324/409205648497
```

General FizzBuzz

```
class FizzBuzz(schema=Hash(<3 Fizz 5 Buzz>...)) {
    method filter(this) {
        var fb = ''
        schema.sort_by {|k,_| k.to_i }.each { |pair|
            fb += (pair[0].to_i `divides` this ? pair[1] : '')
        }
        fb.len > 0 ? fb : this
    }
}

func GeneralFizzBuzz(upto, schema) {
    var ping = FizzBuzz()
    if (nil != schema) {
        ping.schema = schema.to_hash
    }
    (1..upto).map {|i| ping.filter(i) }
}

GeneralFizzBuzz(20, <3 Fizz 5 Buzz 7 Baxx>).each { .say }
```

Output:

```
1
2
Fizz
4
Buzz
Fizz
Baxx
8
Fizz
Buzz
11
Fizz
13
Baxx
FizzBuzz
16
17
Fizz
19
Buzz
```

Generate Chess960 starting position

```
func is_valid_960 (backrank) {
    var king = backrank.index('♚')
    var (rook1, rook2) = backrank.indices_of('♜')...
    king.is_between(rook1, rook2) || return false
    var (bishop1, bishop2) = backrank.indices_of('♝')...
    bishop1+bishop2 -> is_odd
}

func random_960_position(pieces = <♛ ♚ ♜ ♜ ♝ ♝ ♞ ♞>) {
    pieces.shuffle.permutations {|*a|
        return a if is_valid_960(a)
    }
}

say random_960_position().join(' ')
```

Output:

```
♝ ♝ ♜ ♚ ♞ ♛ ♜ ♞
```

Generate lower case ASCII alphabet

```
var arr = 'a'..'z'
say arr.join(' ')
```

Generate random numbers without repeating a value

```
var nums = (1..20).to_a
5.times{ say nums.shuffle.join(" ") }
```

Output:

```
7 16 11 2 8 5 19 1 3 17 10 4 18 6 9 13 15 20 12 14
20 4 18 7 16 2 3 10 5 13 19 17 12 1 6 11 8 15 14 9
2 6 8 18 5 15 1 13 19 17 12 3 4 7 20 16 10 11 9 14
2 18 10 16 12 14 7 13 1 8 15 20 6 17 3 11 5 9 4 19
2 17 14 15 5 13 4 16 11 18 1 10 9 7 6 12 20 3 8 19
```

Generator/Exponential

```
func gen_pow(m) {
    var e = 0;
    func { e++ ** m };
}

func gen_filter(g1, g2) {
    var v2 = g2.run;
    func {
        loop {
            var v1 = g1.run;
            while (v1 > v2) { v2 = g2.run };
            v1 == v2 || return v1;
        }
    }
}

# Create generators.
var squares = gen_pow(2);
var cubes = gen_pow(3);
var squares_without_cubes = gen_filter(squares, cubes);

# Drop 20 values.
20.times { squares_without_cubes() };

# Print 10 values.
var answer = [];
10.times { answer.append(squares_without_cubes()) };
say answer;
```

Output:

```
[529, 576, 625, 676, 784, 841, 900, 961, 1024, 1089]
```

Generic swap

```
func swap(Ref a, Ref b) {
    var tmp = *a
    *a = *b
    *b = tmp
}
```

or:

```
func swap(Ref a, Ref b) {
    (*a, *b) = (*b, *a)
}
```

or:

```
func swap(Ref a, Ref b) {
    [*a, *b] » (b, a)
}
```

The swap functions must be called with variable references.

```
var (x, y) = (1, 2)
swap(\x, \y)
```

Get system command output

Using backticks:

```
var output = `ls`            # `output` is a string
var lines  = `ls`.lines      # `lines` is an array
```

Using pipes:

```
var pipe   = %p(ls)           # same as: Pipe('ls')
var pipe_h = pipe.open_r      # open the pipe for reading
var lines  = []               # will store the lines of the output
pipe_h.each { |line| lines << line }
```

Getting the number of decimals

```
func number_of_decimals(n, limit = 1e5) {
    var prec = Num(Num!PREC)>>2
    var prev = ''

    n = Number(n) if !n.kind_of(Number)

    loop {
        var str = n.as_dec(prec)

        if (prev == str) {
            return (str.contains('.') ? str.substr(str.index('.')+1).len : 0)
        }

        prev = str
        prec *= 2
        return Inf if (prec > limit)
    }
}

var list = [
    9, 12.345, "12.3450", "12.345e53",
    12.34555555555555555555, 0.1234567890987654321,
    Num.pi, 1/3, 1.5**63
]

list.each {|n|
    var c = number_of_decimals(n)
    say "Number of decimals: #{'%3s' % c}  Number: #{n}"
}
```

Output:

```
Number of decimals:   0  Number: 9
Number of decimals:   3  Number: 12.345
Number of decimals:   3  Number: 12.3450
Number of decimals:   0  Number: 12.345e53
Number of decimals:  20  Number: 12.34555555555555555555
Number of decimals:  19  Number: 0.1234567890987654321
Number of decimals: 188  Number: 3.14159265358979323846264338327950288419716939938
Number of decimals: Inf  Number: 0.33333333333333333333333333333333333333333333333333
Number of decimals:  63  Number: 124093581919.64894769782737365038018800822428038254175148 9
```

Globally replace text in several files

```
var names = %w(
                a.txt
                b.txt
                c.txt
              )

names.map{ File(_) }.each { |file|
    say file.edit { |line|
        line.gsub("Goodbye London!", "Hello New York!")
    }
}
```

Gray code

```
func bin2gray(n) {
    n ^ (n >> 1)
}

func gray2bin(num) {
    var bin = num
    while (num >>= 1) { bin ^= num }
    return bin
}

{ |i|
    var gr = bin2gray(i)
    printf("%d\t%b\t%b\t%b\n", i, i, gr, gray2bin(gr))
} << ^32
```

Output:

```
0       0       0       0
1       1       1       1
2       10      11      10
3       11      10      11
4       100     110     100
5       101     111     101
6       110     101     110
7       111     100     111
8       1000    1100    1000
9       1001    1101    1001
10      1010    1111    1010
11      1011    1110    1011
12      1100    1010    1100
13      1101    1011    1101
14      1110    1001    1110
15      1111    1000    1111
16      10000   11000   10000
17      10001   11001   10001
18      10010   11011   10010
19      10011   11010   10011
20      10100   11110   10100
21      10101   11111   10101
22      10110   11101   10110
23      10111   11100   10111
24      11000   10100   11000
25      11001   10101   11001
26      11010   10111   11010
27      11011   10110   11011
28      11100   10010   11100
29      11101   10011   11101
30      11110   10001   11110
31      11111   10000   11111
```

Grayscale image

```
require('Image::Imlib2')

func tograyscale(img) {
    var (width, height) = (img.width, img.height)
    var gimg = %O<Image::Imlib2>.new(width, height)
    for y,x in (^height ~X ^width) {
        var (r, g, b) = img.query_pixel(x, y)
        var gray = int(0.2126*r + 0.7152*g + 0.0722*b)
        gimg.set_color(gray, gray, gray, 255)
        gimg.draw_point(x, y)
    }
    return gimg
}

var (input='input.png', output='output.png') = ARGV...
var image = %O<Image::Imlib2>.load(input)
var gscale = tograyscale(image)
gscale.set_quality(80)
gscale.save(output)
```

Greatest common divisor

Built-in:

```
var arr = [100, 1_000, 10_000, 20]
say Math.gcd(arr...)
```

Recursive Euclid algorithm:

```
func gcd(a, b) {
    b==0 ? a.abs : gcd(b, a % b)
}
```

Greatest element of a list

max method returns the greatest element in a list. It works only if the array's elements have the same type (e.g.: strings, numbers).

```
values.max
```

Greatest subsequential sum

```
func maxsubseq(*a) {
    var (start, end, sum, maxsum) = (-1, -1, 0, 0)
    a.each_kv { |i, x|
        sum += x
        if (maxsum < sum) {
            maxsum = sum
            end = i
        }
        elsif (sum < 0) {
            sum = 0
            start = i
        }
    }
    a.ft(start+1, end)
}

say maxsubseq(-1, -2,  3,  5,  6, -2, -1,  4, -4,  2, -1)
say maxsubseq(-2, -2, -1,  3,  5,  6, -1,  4, -4,  2, -1)
say maxsubseq(-2, -2, -1, -3, -5, -6, -1, -4, -4, -2, -1)
```

Output:

```
[3, 5, 6, -2, -1, 4]
[3, 5, 6, -1, 4]
[]
```

Guess the number

```
var n = pick(1..10)
print 'Guess the number: '
while (n != read(Number).int) {
    print 'Wrong! Guess again: '
}
say 'Well guessed!'
```

Guess the number/With feedback

```
var number = pick(1..10)
say "Guess the number between 1 and 10"

loop {
    given(var n = read("> ", Number).to_i) {
        when (number)     { say "You guessed it."; break }
        case (n < number) { say "Too low" }
        default           { say "Too high" }
    }
}
```

Guess the number/With feedback (player)

```
var min = 1
var max = 99
var tries = 0
var guess = pick(min..max)

print <<"EOT".chomp
\n=>> Think of a number between #{min} and #{max} and I'll guess it!\n
Press <ENTER> when are you ready...
EOT

STDIN.readline

loop {
    print <<-EOT.chomp
    \n=>> My guess is: #{guess} Is your number higher, lower, or equal? (h/l/e)
    >#{' '}
    EOT

    ++tries
    given (STDIN.readline) {
        case (max <= min) {
            say "\nI give up..."
            break
        }
        when (/^h/i) {
            min = guess+1
        }
        when (/^l/i) {
            max = guess
        }
        when (/^e/i) {
            say "\nI knew it! It took me only #{tries} tries."
            break
        }
        default {
            say "error: invalid score"
            next
        }
    }

    guess = ((min+max) // 2)
}
```

GUI/Maximum window dimensions

Using the Tk library:

```
require('Tk')

func max_window_size() -> (Number, Number) {
    %O<MainWindow>.new.maxsize
}

var (width, height) = max_window_size()
say (width, 'x', height)
```

Output:

```
1905x1050
```

Hailstone sequence

```
func hailstone (n) {
    var sequence = [n]
    while (n > 1) {
        sequence << (
            n.is_even ? n.div!(2)
                      : n.mul!(3).add!(1)
        )
    }
    return(sequence)
}

# The hailstone sequence for the number 27
var arr = hailstone(var nr = 27)
say "#{nr}: #{arr.first(4)} ... #{arr.last(4)} (#{arr.len})"

# The longest hailstone sequence for a number less than 100,000
var h = [0, 0]
for i (1 .. 99_999) {
    (var l = hailstone(i).len) > h[1] && (
        h = [i, l]
    )
}

printf("%d: (%d)\n", h...)
```

Hamming numbers

```
func ham_gen {
    var s = [[1], [1], [1]]
    var m = [2, 3, 5]

    func {
        var n = [s[0][0], s[1][0], s[2][0]].min
        { |i|
            s[i].shift if (s[i][0] == n)
            s[i].append(n * m[i])
        } << ^3
        return n
    }
}

var h = ham_gen()

var i = 20;
say i.of { h() }.join(' ')

{ h() } << (i+1 ..^ 1691)
say h()
```

Output:

```
1 2 3 4 5 6 8 9 10 12 15 16 18 20 24 25 27 30 32 36
2125764000
```

Handle a signal

```
var start = Time.sec

Sig.INT {
    say "Ran for #{Time.sec - start} seconds."
    Sys.exit
}

{ |i|
    say i
    Sys.sleep(0.5)
} * Inf
```

Output:

```
1
2
3
4
^CRan for 2 seconds.
```

Happy numbers

```
func happy(n) is cached {
    static seen = Hash()

    return true  if n.is_one
    return false if seen.exists(n)

    seen{n} = 1
    happy(n.digits.sum { _*_ })
}

say happy.first(10)
```

Output:

```
[1, 7, 10, 13, 19, 23, 28, 31]
```

Harshad or Niven series

```
func harshad() {
    var n = 0
    {
        ++n while !n.digits.sum.divides(n)
        n
    }
}

var iter = harshad()
say 20.of { iter.run }

var n
do {
    n = iter.run
} while (n <= 1000)

say n
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 18, 20, 21, 24, 27, 30, 36, 40, 42]
1002
```

Hash from two arrays

```
var keys = %w(a b c)
var vals = [1, 2, 3]

var hash = Hash()
hash{keys...} = vals...
say hash
```

Hash join

```
func hashJoin(table1, index1, table2, index2) {
    var a = []
    var h = Hash()

    # hash phase
    table1.each { |s|
        h{s[index1]} := [] << s
    }

    # join phase
    table2.each { |r|
        a += h{r[index2]}.map{[_,r]}
    }

    return a
}

var t1  = [[27, "Jonah"],
           [18, "Alan"],
           [28, "Glory"],
           [18, "Popeye"],
           [28, "Alan"]]

var t2  = [["Jonah", "Whales"],
           ["Jonah", "Spiders"],
           ["Alan", "Ghosts"],
           ["Alan", "Zombies"],
           ["Glory", "Buffy"]]

hashJoin(t1, 1, t2, 0).each { .say }
```

Output:

```
[[27, 'Jonah'], ['Jonah', 'Whales']]
[[27, 'Jonah'], ['Jonah', 'Spiders']]
[[18, 'Alan'], ['Alan', 'Ghosts']]
[[28, 'Alan'], ['Alan', 'Ghosts']]
[[18, 'Alan'], ['Alan', 'Zombies']]
[[28, 'Alan'], ['Alan', 'Zombies']]
[[28, 'Glory'], ['Glory', 'Buffy']]
```

Haversine formula

```
class EarthPoint(lat, lon) {

    const earth_radius = 6371       # mean earth radius
    const radian_ratio = Num.pi/180

    # accessors for radians
    method latR { self.lat * radian_ratio }
    method lonR { self.lon * radian_ratio }

    method haversine_dist(EarthPoint p) {
        var arc = EarthPoint(
              self.lat - p.lat,
              self.lon - p.lon,
        )

        var a = Math.sum(
                  (arc.latR / 2).sin**2,
                  (arc.lonR / 2).sin**2 *
                    self.latR.cos * p.latR.cos
                )

        earth_radius * a.sqrt.asin * 2
    }
}

var BNA = EarthPoint.new(lat: 36.12, lon: -86.67)
var LAX = EarthPoint.new(lat: 33.94, lon: -118.4)

say BNA.haversine_dist(LAX)   #=> 2886.444442837983299747157823945746716...
```

Hello world/Graphical

Tk:

```
var tk = require('Tk')
var main = %O<MainWindow>.new
main.Button(
    '-text'    => 'Goodbye, World!',
    '-command' => 'exit',
).pack
tk.MainLoop
```

Gtk2:

```
var gtk2 = require('Gtk2') -> init

var window = %O<Gtk2::Window>.new
var label  = %O<Gtk2::Label>.new('Goodbye, World!')

window.set_title('Goodbye, World!')
window.signal_connect(destroy => { gtk2.main_quit })

window.add(label)
window.show_all

gtk2.main
```

Hello world/Line printer

```
Sys.open(\var fh, '>', '/dev/lp0') \
    && fh.say("Hello World!")      \
    && fh.close
```

Hello world/Newline omission

```
print "Goodbye, World!"
```

or:

```
"%s".printf("Goodbye, World!")
```

Hello world/Standard error

```
STDERR.say("Goodbye, World!")
```

Hello world/Text

```
„Hello world!”.say
```

Or:

```
say „Hello world!”
```

Hello world/Web server

Using the low-level *Socket* object:

```
var port = 8080;
var protocol = Socket.getprotobyname("tcp");

var sock = (Socket.open(Socket.PF_INET, Socket.SOCK_STREAM, protocol) || die "couldn't open a socket:
#{$!}");
  # PF_INET to indicate that this socket will connect to the internet domain
  # SOCK_STREAM indicates a TCP stream, SOCK_DGRAM would indicate UDP communication

sock.setsockopt(Socket.SOL_SOCKET, Socket.SO_REUSEADDR, 1) || die "couldn't set socket options: #
{$!}";
  # SOL_SOCKET to indicate that we are setting an option on the socket instead of the protocol
  # mark the socket reusable

sock.bind(Socket.sockaddr_in(port, Socket.INADDR_ANY)) || die "couldn't bind socket to port #{port}:
#{$!}";
  # bind our socket to $port, allowing any IP to connect

sock.listen(Socket.SOMAXCONN) || die "couldn't listen to port #{port}: #{$!}";
  # start listening for incoming connections

while (var client = sock.accept) {
  client.print ("HTTP/1.1 200 OK\r\n" +
               "Content-Type: text/html; charset=UTF-8\r\n\r\n" +
               "<html><head><title>Goodbye, world!</title></head>" +
               "<body>Goodbye, world!</body></html>\r\n");
  client.close;
}
```

A more friendly interface, using the *IO::Socket::INET* library:

```
var inet = require('IO::Socket::INET');

var sock = inet.new( LocalAddr => "127.0.0.1:8080",
                     Listen    => 1,
                     Reuse     => 1,
           );

while (var client = sock.accept) {
    client.print ("HTTP/1.1 200 OK\r\n" +
                 "Content-Type: text/html; charset=UTF-8\r\n\r\n" +
                 "<html><head><title>Goodbye, world!</title></head>" +
                 "<body>Goodbye, world!</body></html>\r\n");
    client.close;
}
```

Here document

There must not be a space between the "<<" and the token string. When the token string is double-quoted ("") or not quoted, the content will be interpolated like a double-quoted string:

```
var text = <<"EOF"
a = #{1+2}
b = #{3+4}
EOF
```

If single quotes are used, then the here document will not support interpolation, like a normal single-quoted string:

```
var x = <<'FOO'
No
#{interpolation}
here
FOO
```

The here document does not start immediately at the "<<END" token -- it starts on the next line. The "<<END" is actually an expression, whose value will be substituted by the contents of the here document. To further illustrate this fact, we can use the "<<END" inside a complex, nested expression:

```
say (<<EOF + "lamb")
Mary had
  a little
EOF
```

which is equivalent with:

```
say (<<EOF
Mary had
  a little
EOF
+ "lamb");
```

Heronian triangles

```
class Triangle(a, b, c) {

  has (sides, perimeter, area)

  method init {
    sides = [a, b, c].sort
    perimeter = [a, b, c].sum
    var s = (perimeter / 2)
    area = sqrt(s * (s - a) * (s - b) * (s - c))
  }

  method is_valid(a,b,c) {
    var (short, middle, long) = [a, b, c].sort...
    (short + middle) > long
  }

  method is_heronian {
    area == area.to_i
  }

  method <=>(other) {
    [area, perimeter, sides] <=> [other.area, other.perimeter, other.sides]
  }

  method to_s {
    "%-11s%6d%8.1f" % (sides.join('x'), perimeter, area)
  }
}

var (max, area) = (200, 210)
var prim_triangles = []

for a in (1..max) {
  for b in (a..max) {
    for c in (b..max) {
      next if (Math.gcd(a, b, c) > 1)
      prim_triangles << Triangle(a, b, c) if Triangle.is_valid(a, b, c)
    }
  }
}

var sorted = prim_triangles.grep{.is_heronian}.sort

say "Primitive heronian triangles with sides upto #{max}: #{sorted.size}"
say "\nsides       perim.   area"
say sorted.first(10).join("\n")
say "\nTriangles with an area of: #{area}"
sorted.each{|tr| say tr if (tr.area == area)}
```

Output:

```
Primitive heronian triangles with sides upto 200: 517

sides       perim.   area
3x4x5          12     6.0
5x5x6          16    12.0
5x5x8          18    12.0
4x13x15        32    24.0
5x12x13        30    30.0
9x10x17        36    36.0
3x25x26        54    36.0
7x15x20        42    42.0
10x13x13       36    60.0
8x15x17        40    60.0

Triangles with an area of: 210
17x25x28       70   210.0
20x21x29       70   210.0
12x35x37       84   210.0
17x28x39       84   210.0
7x65x68       140   210.0
3x148x149     300   210.0
```

Hickerson series of almost integers

```
func h(n) {
    n! / (2 * pow(2.log, n+1))
}

{ |n|
    "h(%2d) = %22s is%s almost an integer.\n".printf(
        n, var hn = h(n).round(-3), hn.to_s ~~ /\.[09]/ ? '' : ' NOT')
} << 1..17
```

Output:

```
h( 1) =                  1.041 is almost an integer.
h( 2) =                  3.003 is almost an integer.
h( 3) =                 12.996 is almost an integer.
h( 4) =                 74.999 is almost an integer.
h( 5) =                541.002 is almost an integer.
h( 6) =               4683.001 is almost an integer.
h( 7) =              47292.999 is almost an integer.
h( 8) =             545834.998 is almost an integer.
h( 9) =            7087261.002 is almost an integer.
h(10) =          102247563.005 is almost an integer.
h(11) =         1622632572.998 is almost an integer.
h(12) =        28091567594.982 is almost an integer.
h(13) =       526858348381.001 is almost an integer.
h(14) =     10641342970443.085 is almost an integer.
h(15) =    230283190977853.037 is almost an integer.
h(16) =   5315654681981354.513 is NOT almost an integer.
h(17) = 130370767029135900.458 is NOT almost an integer.
```

Higher-order functions

```
func first(f) {
  return f()
}

func second {
  return "second"
}

say first(second)               # => "second"
say first(func { "third" })     # => "third"
```

Hilbert curve

Generic implementation of the Lindenmayer system:

```
require('Image::Magick')

class Turtle(
    x      = 500,
    y      = 500,
    angle  = 0,
    scale  = 1,
    mirror = 1,
    xoff   = 0,
    yoff   = 0,
    color  = 'black',
) {

    has im = %O<Image::Magick>.new(size => "#{x}x#{y}")

    method init {
        angle.deg2rad!
        im.ReadImage('canvas:white')
    }

    method forward(r) {
        var (newx, newy) = (x + r*sin(angle), y + r*-cos(angle))

        im.Draw(
            primitive => 'line',
            points    => join(' ',
                          round(x    * scale + xoff),
                          round(y    * scale + yoff),
                          round(newx * scale + xoff),
                          round(newy * scale + yoff),
                      ),
            stroke      => color,
            strokewidth => 1,
        )

        (x, y) = (newx, newy)
    }

    method save_as(filename) {
        im.Write(filename)
    }

    method turn(theta) {
        angle += theta*mirror
    }
```

```
    method state {
        [x, y, angle, mirror]
    }

    method setstate(state) {
        (x, y, angle, mirror) = state...
    }

    method mirror {
        mirror.neg!
    }
}

class LSystem(
    angle  = 90,
    scale  = 1,
    xoff   = 0,
    yoff   = 0,
    len    = 5,
    color  = 'black',
    width  = 500,
    height = 500,
    turn   = 0,
) {
    method execute(string, repetitions, filename, rules) {

        var theta  = angle.deg2rad
        var turtle = Turtle(
            x:     width,
            y:     height,
            angle: turn,
            scale: scale,
            color: color,
            xoff:  xoff,
            yoff:  yoff,
        )

        var stack = []
        var table = Hash(
            '+' => { turtle.turn(theta) },
            '-' => { turtle.turn(-theta) },
            ':' => { turtle.mirror },
            '[' => { stack.push(turtle.state) },
            ']' => { turtle.setstate(stack.pop) },
        )

        repetitions.times {
            string.gsub!(/(.)/, {|c| rules{c} \\ c })
        }

        string.each_char { |c|
            if (table.contains(c)) {
                table{c}.run
            }
            elsif (c.is_uppercase) {
                turtle.forward(len)
            }
        }

        turtle.save_as(filename)
    }
}
```

Generating the Hilbert curve:

```
var rules = Hash(
    a => '-bF+aFa+Fb-',
    b => '+aF-bFb-Fa+',
)

var lsys = LSystem(
    width:  600,
    height: 600,

    xoff: -50,
    yoff: -50,

    len:   8,
    angle: 90,
    color: 'dark green',
)

lsys.execute('a', 6, "hilbert_curve.png", rules)
```

Output image: Hilbert curve

History variables

Implemented as a class:

```
class HistoryVar(v) {

    has history = []
    has variable = v

    method ≔(value) {
        history << variable
        variable = value
    }

    method to_s {
        "#{variable}"
    }

    method AUTOLOAD(_, name, *args) {
        variable.(name)(args...)
    }
}

var foo = HistoryVar(0)

foo ≔ 1
foo ≔ 2
foo ≔ foo+3
foo ≔ 42

say "History: #{foo.history}"
say "Current value: #{foo}"
```

Output:

```
History: [0, 1, 2, 5]
Current value: 42
```

Hofstadter-Conway $10,000 sequence

```
class HofstadterConway10000 {
  has sequence = [nil, 1, 1]
  method term(n {.is_pos}) {
    var a = sequence
    {|i| a[i] = a[a[i-1]]+a[i-a[i-1]] } << a.len..n
    a[n]
  }
}

var hc = HofstadterConway10000()

var mallows = nil
for i in (1..19) {
  var j = i+1
  var (max_n, max_v) = (-1, -1)
  for n in (1<<i .. 1<<j) {
    var v = (hc.term(n) / n)
    (max_n, max_v) = (n, v) if (v > max_v)
    mallows = n if (v >= 0.55)
  }
  say ("maximum between 2^%2d and 2^%2d occurs at%7d: %.8f" % (i, j, max_n, max_v))
}

say "the mallows number is #{mallows}"
```

Output:

```
maximum between 2^ 1 and 2^ 2 occurs at      3: 0.66666667
maximum between 2^ 2 and 2^ 3 occurs at      6: 0.66666667
maximum between 2^ 3 and 2^ 4 occurs at     11: 0.63636364
maximum between 2^ 4 and 2^ 5 occurs at     23: 0.60869565
maximum between 2^ 5 and 2^ 6 occurs at     44: 0.59090909
maximum between 2^ 6 and 2^ 7 occurs at     92: 0.57608696
maximum between 2^ 7 and 2^ 8 occurs at    178: 0.56741573
maximum between 2^ 8 and 2^ 9 occurs at    370: 0.55945946
maximum between 2^ 9 and 2^10 occurs at    719: 0.55493741
maximum between 2^10 and 2^11 occurs at   1487: 0.55010087
maximum between 2^11 and 2^12 occurs at   2897: 0.54746289
maximum between 2^12 and 2^13 occurs at   5969: 0.54414475
maximum between 2^13 and 2^14 occurs at  11651: 0.54244271
maximum between 2^14 and 2^15 occurs at  22223: 0.54007110
maximum between 2^15 and 2^16 occurs at  45083: 0.53878402
maximum between 2^16 and 2^17 occurs at  89516: 0.53704366
maximum between 2^17 and 2^18 occurs at 181385: 0.53602007
maximum between 2^18 and 2^19 occurs at 353683: 0.53464543
maximum between 2^19 and 2^20 occurs at 722589: 0.53377923
the mallows number is 1489
```

Hofstadter Figure-Figure sequences

```
var r = [nil, 1]
var s = [nil, 2]

func ffsr(n) {
  while(r.end < n) {
    r << s[r.end]+r[-1]
    s << [(s[-1]+1 .. r[-1]-1)..., r[-1]+1].grep{ s[-1] < _ }...
  }
  return n
}

func ffr(n) { r[ffsr(n)] }
func ffs(n) { s[ffsr(n)] }

printf("  i: R(i) S(i)\n")
printf("==============\n")
{ |i|
    printf("%3d:  %3d  %3d\n", i, ffr(i), ffs(i))
} << 1..10
printf("\nR(40)=%3d S(960)=%3d R(41)=%3d\n", ffr(40), ffs(960), ffr(41))

var seen = Hash()

{|i| seen{ffr(i)} := 0 ++ } << 1..40
{|i| seen{ffs(i)} := 0 ++ } << 1..960

if (seen.count {|k,v| (k.to_i >= 1) && (k.to_i <= 1000) && (v == 1) } == 1000) {
    say "All occured exactly once."
}
else {
    var missed = { !seen.has_key(_) }.grep(1..1000)
    var dupped = seen.grep { |_, v| v > 1 }.keys.sort
    say "These were missed: #{missed}"
    say "These were duplicated: #{dupped}"
}
```

Output:

```
  i: R(i) S(i)
==============
  1:    1    2
  2:    3    4
  3:    7    5
  4:   12    6
  5:   18    8
  6:   26    9
  7:   35   10
  8:   45   11
  9:   56   13
 10:   69   14

R(40)=982 S(960)=1000 R(41)=1030
All occured exactly once.
```

Hofstadter Q sequence

Using a memoized function:

```
func Q(n) is cached {
    n <= 2 ? 1
           : Q(n - Q(n-1))+Q(n-Q(n-2))
}

say "First 10 terms: #{ {|n| Q(n) }.map(1..10) }"
say "Term 1000: #{Q(1000)}"
say "Terms less than preceding in first 100k: #{2..100000->count{|i|Q(i)<Q(i-1)}}"
```

Using an array:

```
var Q = [0, 1, 1]
100_000.times {
    Q << (Q[-Q[-1]] + Q[-Q[-2]])
}

say "First 10 terms: #{Q.ft(1, 10)}"
say "Term 1000: #{Q[1000]}"
say "Terms less than preceding in first 100k: #{2..100000->count{|i|Q[i]<Q[i-1]}}"
```

Output:

```
First 10 terms: [1, 1, 2, 3, 3, 4, 5, 5, 6, 6]
Term 1000: 502
Terms less than preceding in first 100k: 49798
```

Holidays related to Easter

```
require('Date::Calc')

var abbr = < Nil Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec >

var holidays = [
    [Easter    => 0],
    [Ascension => 39],
    [Pentecost => 49],
    [Trinity   => 56],
    [Corpus    => 60],
]

func easter(year) {
    var ay = (year % 19)
    var by = (year // 100)
    var cy = (year % 100)
    var dy = (by // 4)
    var ey = (by % 4)
    var fy = ((by + 8) // 25)
    var gy = ((by - fy + 1) // 3)
    var hy = ((19*ay + by - dy - gy + 15) % 30)
    var iy = (cy // 4)
    var ky = (cy % 4)
    var ly = ((32 + 2*ey + 2*iy - hy - ky) % 7)
    var md = (hy + ly - 7*((ay + 11*hy + 22*ly) // 451) + 114)
    var month = (md // 31)
    var day = (md % 31 + 1)
    return(month, day)
}

func cholidays(year) {
    var (emon, eday) = easter(year)
    printf("%4s: ", year)
    say gather {
        holidays.each { |holiday|
            var (_, mo, da) = %S<Date::Calc>.Add_Delta_Days(year, emon, eday, holiday[1])
            take("#{holiday[0]}: #{'%02d' % da} #{abbr[mo]}")
        }
    }.join(', ')
}

for year in (400..2100 `by` 100, 2010..2020) {
    cholidays(year)
}
```

Home primes

```
for n in (2..20, 65) {

    var steps = []
    var orig = n

    for (var f = n.factor; true; f = n.factor) {
        steps << f
        n = Num(f.join)
        break if n.is_prime
    }

    say ("HP(#{orig}) = ", steps.map { .join('_') }.join(' -> '))
}
```

Output:

```
HP(2) = 2
HP(3) = 3
HP(4) = 2_2 -> 2_11
HP(5) = 5
HP(6) = 2_3
HP(7) = 7
HP(8) = 2_2_2 -> 2_3_37 -> 3_19_41 -> 3_3_3_7_13_13 -> 3_11123771 -> 7_149_317_941 -> 229_31219729 -> 1
HP(9) = 3_3 -> 3_11
HP(10) = 2_5 -> 5_5 -> 5_11 -> 7_73
HP(11) = 11
HP(12) = 2_2_3
HP(13) = 13
HP(14) = 2_7 -> 3_3_3 -> 3_3_37 -> 47_71 -> 13_367
HP(15) = 3_5 -> 5_7 -> 3_19 -> 11_29
HP(16) = 2_2_2_2 -> 2_11_101 -> 3_11_6397 -> 3_163_6373
HP(17) = 17
HP(18) = 2_3_3
HP(19) = 19
HP(20) = 2_2_5 -> 3_3_5_5 -> 5_11_61 -> 11_4651 -> 3_3_12739 -> 17_194867 -> 19_41_22073 -> 709_273797
HP(65) = 5_13 -> 3_3_3_19 -> 11_13_233 -> 11_101203 -> 3_3_23_53629 -> 3_3_1523_24247 -> 3_3_3_7_47_373
```

Honeycombs

```
require('Tk')

class Honeycombs(
    Number size   = 36,
    Array letters = @('A' .. 'Z').shuffle.first(20),
) {

    define tk   = %S<Tk>
    has changed = Hash()

    func altitude(n) {
        sqrt(3/4) * n
    }

    method polygon_coordinates(x, y, size) {
        var alt = altitude(size)
        return (x - size,       y,
                x - size/2, y - alt,
                x + size/2, y - alt,
                x + size,       y
```

```
                x + size,       y,
                x + size/2, y + alt,
                x - size/2, y + alt,
               );
    }

    method change(canvas, id, letter_id) {
        return {
            canvas.itemconfigure(id,        '-fill' => 'magenta')
            canvas.itemconfigure(letter_id, '-fill' => 'black')
            changed{id} = true

            if (20 == changed.len) {
                say "All letters pressed."
                canvas.MainWindow.after(10, { tk.exit })
            }
        }
    }

    method comb(canvas, fromx, fromy, size, count) {
        for x,y in (
            RangeNum(fromx, 3*count*size - 1,          3*size) ~X
            RangeNum(fromy,     7.5*size - 1, 2*altitude(size))
        ) {
            var id = canvas.createPolygon(
                              self.polygon_coordinates(x, y, size),
                              '-outline' => 'black',
                              '-fill'    => 'yellow',
                              '-width'   => 2,
                             )
            var letter = letters.shift
            var letter_id = canvas.createText(x, y,
                                '-fill' => 'red',
                                '-text' => letter,
                                '-font' => "{sans} #{size * 0.9}",
                               )
            canvas.MainWindow.bind('all', letter.lc,
                                   self.change(canvas, id, letter_id))
            [id, letter_id].each { |b|
                canvas.bind(b, '<Button-1>',
                                   self.change(canvas, id, letter_id))
            }
        }
    }

    method display(title) {
        {
            var mw     = %s'MainWindow'.new('-title' => title)
            var canvas = mw.Canvas('-width'  => 8*size,
                                   '-height' => 8*size).pack

            self.comb(canvas, size,       size,                  size, 3)
            self.comb(canvas, size * 2.5, size + altitude(size), size, 2)

            var btn = mw.Button('-text'      => 'Quit',
                                '-underline' => 0,
                                '-command'   => { tk.exit },
                               ).pack
            mw.bind('<Alt-q>', { btn.invoke })
            tk.MainLoop()
        }.fork
    }
}

Honeycombs().display(title: 'Honeycombs')
```

Horizontal sundial calculations

```
var latitude  = read('Enter latitude       => ', Number)
var longitude = read('Enter longitude      => ', Number)
var meridian  = read('Enter legal meridian => ', Number)

var lat_sin = latitude.deg2rad.sin
var offset = (meridian - longitude)

say('Sine of latitude: ', "%.4f" % lat_sin)
say('Longitude offset: ', offset)
say('=' * 48)
say(' Hour  : Sun hour angle° : Dial hour line angle°')

for hour in range(-6, 6) {
    var sun_deg  = (15*hour + offset)
    var line_deg = rad2deg(
        atan2(
            sin(deg2rad(sun_deg)) * lat_sin,
            cos(deg2rad(sun_deg))
        )
    )
    printf("%2d %s      %7.3f             %7.3f\n",
      (hour + 12) % 12 || 12, (hour < 0 ? 'AM' : 'PM'), sun_deg, line_deg)
}
```

Output:

```
Enter latitude       => -4.95
Enter longitude      => -150.5
Enter legal meridian => -150
Sine of latitude: -0.0863
Longitude offset: 0.5
================================================
 Hour  : Sun hour angle° : Dial hour line angle°
 6 AM      -89.500              84.225
 7 AM      -74.500              17.283
 8 AM      -59.500               8.334
 9 AM      -44.500               4.847
10 AM      -29.500               2.795
11 AM      -14.500               1.278
12 PM        0.500              -0.043
 1 PM       15.500              -1.371
 2 PM       30.500              -2.910
 3 PM       45.500              -5.018
 4 PM       60.500              -8.671
 5 PM       75.500             -18.451
 6 PM       90.500             -95.775
```

Horner's rule for polynomial evaluation

Functional:

```
func horner(coeff, x) {
    coeff.reverse.reduce { |a,b| a*x + b }
}

say horner([-19, 7, -4, 6], 3)   # => 128
```

Recursive:

```
func horner(coeff, x) {
    coeff.len > 0
        ? (coeff[0] + x*horner(coeff.ft(1), x))
        : 0
}

say horner([-19, 7, -4, 6], 3)   # => 128
```

Hostname

```
var sys = frequire('Sys::Hostname')
var host = sys.hostname
```

Or:

```
var host = `hostname`.chomp
```

Hough transform

```
require('Imager')

func hough(im, width=460, height=360) {

    height = 2*floor(height / 2)

    var xsize = im.getwidth
    var ysize = im.getheight

    var ht = %O<Imager>.new(xsize => width, ysize => height)
    var canvas = height.of { width.of(255) }

    ht.box(filled => true, color => 'white')

    var rmax = hypot(xsize, ysize)
    var dr = 2*(rmax / height)
    var dth = (Num.pi / width)

    for y,x in (^ysize ~X ^xsize) {
        var col = im.getpixel(x => x, y => y)
        var (r,g,b) = col.rgba
        (r==255 && g==255 && b==255) && next
        for k in ^width {
            var th = dth*k
            var r = (x*cos(th) + y*sin(th))
            var iry = (height/2 + int(r/dr + 0.5))
            ht.setpixel(x => k, y => iry, color => 3.of(--canvas[iry][k]))
        }
    }

    return ht
}

var img = %O<Imager>.new(file => 'Pentagon.png')
var ht = hough(img)
ht.write(file => 'Hough transform.png')
```

HTTP

Sidef can load and use Perl modules:

```
func get(url) {
    var lwp = (
        try   { require('LWP::UserAgent') }
        catch { warn "'LWP::UserAgent' is not installed!"; return nil }
    )
    var ua = lwp.new(agent => 'Mozilla/5.0')
    if (var resp = ua.get(url); resp.is_success) {
        return resp.decoded_content
    }
    return nil
}

print get("http://rosettacode.org")
```

HTTPS

```
var lwp = require('LWP::UserAgent')    # LWP::Protocol::https is needed
var url = 'https://rosettacode.org'

var ua = lwp.new(
    agent    => 'Mozilla/5.0',
    ssl_opts => Hash(verify_hostname => 1),
)

var resp = ua.get(url)
resp.is_success || die "Failed to GET #{url}: #{resp.status_line}"
print resp.decoded_content
```

HTTPS/Authenticated

```
require('WWW::Mechanize')

var mech = %O<WWW::Mechanize>.new(
    cookie_jar => Hash(),
    agent => 'Mozilla/5.0',
    show_progress => 1,
)

mech.get('https://login.yahoo.com/')
mech.submit_form(
    form_id => 'mbr-login-form',   # form id
    fields => Hash(
        'login'  => 'XXXXXX',
        'passwd' => 'YYYYYY',
))
```

Huffman coding

```
func walk(n, s, h) {
    if (n.contains(:a)) {
        h{n{:a}} = s
        say "#{n{:a}}: #{s}"
        return nil
    }
    walk(n{:0}, s+'0', h)
    walk(n{:1}, s+'1', h)
}

func make_tree(text) {
    var letters = Hash()
    text.each { |c| letters{c} := 0 ++ }
    var nodes = letters.keys.map { |l|
        Hash(a => l, freq => letters{l})
    }

    var n = Hash()
    while (nodes.sort_by!{|c| c{:freq} }.len > 1) {
        n = Hash(:0 => nodes.shift, :1 => nodes.shift)
        n{:freq} = (n{:0}{:freq} + n{:1}{:freq})
        nodes.append(n)
    }

    walk(n, "", n{:tree} = Hash())
    return n
}

func encode(s, t) {
    t = t{:tree}
    s.chars.map{|c| t{c} }.join
}

func decode (enc, tree) {
    var n = tree
    var out = ""

    enc.each {|bit|
        n = n{bit}
        if (n.contains(:a)) {
            out += n{:a}
            n = tree
        }
    }

    return out
}

var text = "this is an example for huffman encoding"
var tree = make_tree(text)
var enc = encode(text, tree)

say enc
say decode(enc, tree)
```

Output:

```
n: 000
s: 0010
o: 0011
h: 0100
l: 01010
g: 01011
x: 01100
c: 01101
d: 01110
u: 01111
p: 10000
t: 10001
i: 1001
 : 101
f: 1100
a: 1101
e: 1110
r: 11110
m: 11111
1000101001001001010110010010101110100010111100110011011111110000010101110101110000111111010101000111111
this is an example for huffman encoding
```

Humble numbers

```
func smooth_generator(primes) {

    var s = primes.len.of { [1] }

    {
        var n = s.map { .first }.min
        { |i|
            s[i].shift if (s[i][0] == n)
            s[i] << (n * primes[i])
        } * primes.len
        n
    }
}

with (smooth_generator([2,3,5,7])) {|g|
    say 50.of { g.run }.join(' ')
}

say "\nThe digit counts of humble numbers"
say '='*35

with (smooth_generator([2,3,5,7])) {|g|
    for (var(d=1,c=0); d <= 20; ++c) {
        var n = g.run
        n.len > d || next
        say "#{'%10s'%c.commify}  have  #{'%2d'%d}  digit#{[:s,''][d==1]}"
        (c, d) = (0, n.len)
    }
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10 12 14 15 16 18 20 21 24 25 27 28 30 32 35 36 40 42 45 48 49 50 54 56 60 63 64 70 7

The digit counts of humble numbers
══════════════════════════════════
         9  have   1  digit
        36  have   2  digits
        95  have   3  digits
       197  have   4  digits
       356  have   5  digits
       579  have   6  digits
       882  have   7  digits
     1,272  have   8  digits
     1,767  have   9  digits
     2,381  have  10  digits
     3,113  have  11  digits
     3,984  have  12  digits
     5,002  have  13  digits
     6,187  have  14  digits
     7,545  have  15  digits
     9,081  have  16  digits
    10,815  have  17  digits
    12,759  have  18  digits
    14,927  have  19  digits
    17,323  have  20  digits
```

IBAN

```
func valid_iban(iban) {
  static len = Hash(
    AD=>24, AE=>23, AL=>28, AO=>25, AT=>20, AZ=>28, BA=>20, BE=>16, BF=>27,
    BG=>22, BH=>22, BI=>16, BJ=>28, BR=>29, CG=>27, CH=>21, CI=>28, CM=>27,
    CR=>21, CV=>25, CY=>28, CZ=>24, DE=>22, DK=>18, DO=>28, DZ=>24, EE=>20,
    EG=>27, ES=>24, FI=>18, FO=>18, FR=>27, GA=>27, GB=>22, GE=>22, GI=>23,
    GL=>18, GR=>27, GT=>28, HR=>21, HU=>28, IE=>22, IL=>23, IR=>26, IS=>26,
    IT=>27, JO=>30, KW=>30, KZ=>20, LB=>28, LI=>21, LT=>20, LU=>20, LV=>21,
    MC=>27, MD=>24, ME=>22, MG=>27, MK=>19, ML=>28, MR=>27, MT=>31, MU=>30,
    MZ=>25, NL=>18, NO=>15, PK=>24, PL=>28, PS=>29, PT=>25, QA=>29, RO=>24,
    RS=>22, SA=>24, SE=>24, SI=>19, SK=>24, SM=>27, SN=>28, TN=>24, TR=>26,
    UA=>29, VG=>24,
  )

  # Ensure upper alphanumeric input.
  iban -= /\s+/g
  iban.uc! ~~ /^[0-9A-Z]+\z/ || return false

  # Validate country code against expected length.
  var cc = iban.substr(0, 2)
  iban.len == len{cc} || return false

  # Shift and convert.
  iban.sub!(/(.{4})(.+)/, {|a,b| b+a})
  iban.gsub!(/([A-Z])/,   {|a| a.ord - 55})

  iban.to_i % 97 == 1
}

say valid_iban("GB82 WEST 1234 5698 7654 32") #=> true
say valid_iban("GB82 TEST 1234 5698 7654 32") #=> false
```

Identity matrix

```
func identity_matrix(n) {
    n.of { |i|
        n.of { |j|
            i == j ? 1 : 0
        }
    }
}

for n (ARGV ? ARGV.map{.to_i} : [4, 5, 6]) {
  say "\n#{n}:"
  for row (identity_matrix(n)) {
    say row.join(' ')
  }
}
```

Output:

```
4:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

5:
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1

6:
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 1
```

Imaginary base numbers

```
func base (Number num, Number radix { _ ~~ (-36 .. -2) }, precision = -15) -> String {
    num || return '0'

    var place  = 0
    var result = ''
    var value  = num
    var upper_bound = 1/(-radix + 1)
    var lower_bound = radix*upper_bound

    while (!(lower_bound <= value) || !(value < upper_bound)) {
        value = num/(radix**++place)
    }

    while ((value || (place > 0)) && (place > precision)) {
        var digit = (radix*value - lower_bound -> int)
        value    =  (radix*value - digit)
```

```
        result += '.' if (!place && !result.contains('.'))
        result += ((digit == -radix) ? (digit-1 -> base(-radix) + '0') : digit.base(-radix))
        place--
    }

    return result
}

func base (Number num, Number radix { .re == 0 }, precision = -8) -> String {

    (radix.im.abs ~~ 2..6) || die "Base #{radix} out of range"

    var (re, im)          = (num.re, num.im)
    var (re_wh, re_fr='') = base(re,          -radix.im**2, precision).split('.')...
    var (im_wh, im_fr='') = base(im/radix.im, -radix.im**2, precision).split('.')...

    func zip (String a, String b) {
        var l = ('0' * abs(a.len - b.len))
        chars(a+l) ~Z chars(b+l) -> flat.join.sub(/0+\z/, '') || '0'
    }

    var whole = zip(re_wh.flip, im_wh.flip).flip
    var fraction = zip(im_fr, re_fr)
    fraction == '0' ? whole : "#{whole}.#{fraction}"
}

func parse_base (String str, Number radix { .re == 0 }) -> Number {

    if (str.char(0) == '-') {
        return (-1 * parse_base(str.substr(1), radix))
    }

    var (whole, frac='') = str.split('.')...

    var fraction = frac.chars.map_kv {|k,v|
        Number(v, radix.im**2) * radix**-(k+1)
    }.sum

    fraction += whole.flip.chars.map_kv {|k,v|
        Number(v, radix.im**2) * radix**k
    }.sum

    return fraction
}

var tests = [0, 2i, 1, 2i, 5, 2i, -13, 2i, 9i, 2i, -3i, 2i, 7.75-7.5i, 2i, .25, 2i, # base 2i tests
    5+5i,  2i, 5+5i,  3i, 5+5i,  4i, 5+5i,  5i, 5+5i,  6i, # same value, positive imaginary bases
    5+5i, -2i, 5+5i, -3i, 5+5i, -4i, 5+5i, -5i, 5+5i, -6i, # same value, negative imaginary bases
    227.65625+10.859375i, 4i] # larger test value

tests.each_slice(2, {|v,r|
    var ibase = base(v, r)
    printf("base(%20s, %2si) = %-10s : parse_base(%12s, %2si) = %s\n",
        v, r.im, ibase, "'#{ibase}'", r.im, parse_base(ibase, r).round(-8))
})
```

Output:

```
base(                   0,  2i) = 0          : parse_base(         '0',  2i) = 0
base(                   1,  2i) = 1          : parse_base(         '1',  2i) = 1
base(                   5,  2i) = 10301      : parse_base(     '10301',  2i) = 5
base(                 -13,  2i) = 1030003    : parse_base(   '1030003',  2i) = -13
base(                  9i,  2i) = 103010.2   : parse_base(  '103010.2',  2i) = 9i
base(                -3i,  2i) = 1030.2      : parse_base(    '1030.2',  2i) = -3i
base(           7.75-7.5i,  2i) = 11210.31   : parse_base(  '11210.31',  2i) = 7.75-7.5i
base(                0.25,  2i) = 1.03       : parse_base(      '1.03',  2i) = 0.25
base(                5+5i,  2i) = 10331.2    : parse_base(   '10331.2',  2i) = 5+5i
base(                5+5i,  3i) = 25.3       : parse_base(      '25.3',  3i) = 5+5i
base(                5+5i,  4i) = 25.c       : parse_base(      '25.c',  4i) = 5+5i
base(                5+5i,  5i) = 15         : parse_base(        '15',  5i) = 5+5i
base(                5+5i,  6i) = 15.6       : parse_base(      '15.6',  6i) = 5+5i
base(                5+5i, -2i) = 11321.2    : parse_base(   '11321.2', -2i) = 5+5i
base(                5+5i, -3i) = 1085.6     : parse_base(    '1085.6', -3i) = 5+5i
base(                5+5i, -4i) = 10f5.4     : parse_base(    '10f5.4', -4i) = 5+5i
base(                5+5i, -5i) = 10o5       : parse_base(      '10o5', -5i) = 5+5i
base(                5+5i, -6i) = 5.u        : parse_base(       '5.u', -6i) = 5+5i
base(227.65625+10.859375i,  4i) = 10234.5678 : parse_base('10234.5678',  4i) = 227.65625+10.859375i
```

Implicit type conversion

Since version 3.00, all the number types (int, rat, float and complex) are unified in the *Number* class and all the needed conversions are done implicitly. Methods from other classes also make implicit conversions where possible.

```
> 1+"2"            #=> 3
> "1"+2            #=> 12
> sqrt(-4)         #=> 2i
> ("a" + [1,2])    #=> a[1,2]
> ('ha' * '3')     #=> hahaha
> ('ha' * true)    #=> ha
```

Include a file

Include a file in the current namespace:

```
include('file.sf')
```

Include a file as module (file must exists in *SIDEF_INC* as *Some/Name.sm*):

```
include Some::Name
# variables are available here as: Some::Name::var_name
```

Increment a numerical string

```
say '1234'.inc    #=> '1235'
say '99'.inc      #=> '100'
```

Index finite lists of positive integers

```
func rank(Array arr) {
    Number(arr.join('a'), 11)
}

func unrank(Number n) {
    n.base(11).split('a').map { Num(_) }
}

var l = [1, 2, 3, 10, 100, 987654321]
say l
var n = rank(l)
say n
var l = unrank(n)
say l
```

Output:

```
[1, 2, 3, 10, 100, 987654321]
14307647611639042485573
[1, 2, 3, 10, 100, 987654321]
```

Bijection:

```
func unrank(Number n) {
    n == 1 ? [0]
           : n.base(2).substr(1).split('0', -1).map{.len}
}

func rank(Array x) {
    x.is_empty ? 0
               : Number('1' + x.map { '1' * _ }.join('0'), 2)
}

for x in (0..10) {
    printf("%3d : %-18s: %d\n", x, unrank(x), rank(unrank(x)))
}

say ''
var x = [1, 2, 3, 5, 8]
say "#{x} => #{rank(x)} => #{unrank(rank(x))}"
```

Output:

```
  0 : []               : 0
  1 : [0]              : 1
  2 : [0, 0]           : 2
  3 : [1]              : 3
  4 : [0, 0, 0]        : 4
  5 : [0, 1]           : 5
  6 : [1, 0]           : 6
  7 : [2]              : 7
  8 : [0, 0, 0, 0]     : 8
  9 : [0, 0, 1]        : 9
 10 : [0, 1, 0]        : 10

[1, 2, 3, 5, 8] => 14401279 => [1, 2, 3, 5, 8]
```

Infinity

```
var a = 1.5/0        # Inf
say a.is_inf         # true
say a.is_pos         # true

var b = -1.5/0       # -Inf
say b.is_ninf        # true
say b.is_neg         # true

var inf = Inf
var ninf = -Inf
say (inf == -ninf)   # true
```

Inheritance/Multiple

```
class Camera {}
class MobilePhone {}
class CameraPhone << Camera, MobilePhone {}
```

Inheritance/Single

```
class Animal {}
class Dog << Animal {}
class Cat << Animal {}
class Lab << Dog {}
class Collie << Dog {}
```

Input loop

To read from the standard input, you can use *STDIN* as your *fh*.

```
var file = File(__FILE__)
file.open_r(\var fh, \var err) || die "#{file}: #{err}"

fh.each { |line|              # iterates the lines of the fh
    line.each_word { |word|   # iterates the words of the line
        say word
    }
}
```

Integer comparison

```
var a = read("a: ", Number)
var b = read("b: ", Number)

if (a < b) {
    say 'Lower'
}
elsif (a == b) {
    say 'Equal'
}
elsif (a > b) {
    say 'Greater'
}
```

Output:

```
% sidef numcmp.sf
a: 21
b: 42
Lower
```

Integer overflow

Sidef has unlimited precision integers.

```
var (a, b, c) = (9223372036854775807, 5000000000000000000, 3037000500)
[-(-a - 1), b + b, -a - a, c * c, (-a - 1)/-1].each { say _ }
```

Output:

```
9223372036854775808
10000000000000000000
-18446744073709551614
9223372037000250000
9223372036854775808
```

Integer roots

```
func root(a, b) {
    b < 2 && return(b)
    var (a1, c) = (a-1, 1)
    var f = {|x| (a1*x + b//(x**a1)) // a }
    var d = f(c)
    var e = f(d)
    while (c !~ [d, e]) {
        (c, d, e) = (d, e, f(e))
    }
    [d, e].min
}

say "First 2,001 digits of the square root of two:"
say root(2, 2 * 100**2000)
```

Output:

```
First 2,001 digits of the square root of two:
14142135623730950488016887242096980[...]32952546758516447107578486024636008
```

Integer sequence

No limit:

```
{|i| say i } * Inf
```

Interactive programming (repl)

```
$ sidef -i
>>> func f(s1, s2, sep) { s1 + sep*2 + s2 };
f
>>> f('Rosetta', 'Code', ':')
"Rosetta::Code"
>>>
```

Inverted syntax

```
# Inverted syntax with assignment
var raining = true
[false]»(\var needumbrella)

# Inverted syntax with conditional expressions
if (raining==true) {needumbrella=true}
{needumbrella=true}.if (raining==true)
(needumbrella=true) if (raining==true)
```

Isqrt (integer square root) of X

Built-in:

```
var n = 1234
say n.isqrt
say n.iroot(2)
```

Explicit implementation for the integer k-th root of n:

```
func rootint(n, k=2) {

    return 0 if (n == 0)

    var s = n
    var u = s
    var v = k-1

    loop {
        u = ((v*s + (n // s**v)) // k)
        break if (u >= s)
        s = u
    }

    return s
}

say rootint.map(0..65).join(' ')

for n in (1..73 `by` 2) {
    printf("isqrt(7^%d):\t %s\n", n, rootint(7**n).commify)
}
```

Output:

```
0 1 1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 7 7 7
isqrt(7^1):      2
isqrt(7^3):      18
isqrt(7^5):      129
isqrt(7^7):      907
isqrt(7^9):      6,352
isqrt(7^11):     44,467
isqrt(7^13):     311,269
isqrt(7^15):     2,178,889
isqrt(7^17):     15,252,229
isqrt(7^19):     106,765,608
isqrt(7^21):     747,359,260
isqrt(7^23):     5,231,514,822
isqrt(7^25):     36,620,603,758
isqrt(7^27):     256,344,226,312
isqrt(7^29):     1,794,409,584,184
isqrt(7^31):     12,560,867,089,291
isqrt(7^33):     87,926,069,625,040
isqrt(7^35):     615,482,487,375,282
isqrt(7^37):     4,308,377,411,626,977
isqrt(7^39):     30,158,641,881,388,842
isqrt(7^41):     211,110,493,169,721,897
isqrt(7^43):     1,477,773,452,188,053,281
isqrt(7^45):     10,344,414,165,316,372,973
isqrt(7^47):     72,410,899,157,214,610,812
isqrt(7^49):     506,876,294,100,502,275,687
isqrt(7^51):     3,548,134,058,703,515,929,815
isqrt(7^53):     24,836,938,410,924,611,508,707
isqrt(7^55):     173,858,568,876,472,280,560,953
isqrt(7^57):     1,217,009,982,135,305,963,926,677
isqrt(7^59):     8,519,069,874,947,141,747,486,745
isqrt(7^61):     59,633,489,124,629,992,232,407,216
isqrt(7^63):     417,434,423,872,409,945,626,850,517
isqrt(7^65):     2,922,040,967,106,869,619,387,953,625
isqrt(7^67):     20,454,286,769,748,087,335,715,675,381
isqrt(7^69):     143,180,007,388,236,611,350,009,727,669
isqrt(7^71):     1,002,260,051,717,656,279,450,068,093,686
isqrt(7^73):     7,015,820,362,023,593,956,150,476,655,802
```

Iterated digits squaring

```
func digit_square_sum_iter(n) is cached {

    if ((n == 1) || (n == 89)) {
        return n
    }

    __FUNC__(n.digits.sum { .sqr })
}

say (1..1e6 -> count_by { digit_square_sum_iter(_) == 89 })
```

Output:

```
856929
```

Jacobi symbol

Also built-in as **kronecker(n,k)**.

```
func jacobi(a, n) {

    assert(n > 0,     "#{n} must be positive")
    assert(n.is_odd, "#{n} must be odd")

    var t = 1
    while (a %= n) {
        if (a.is_even) {
            var v = a.valuation(2)
            t *= (-1)**v if (n%8 ~~ [3,5])
            a >>= v
        }
        (a,n) = (n,a)
        t = -t if ([a%4, n%4] == [3,3])
    }

    n==1 ? t : 0
}

for a in (0..50), n in (0..50) {
    assert_eq(jacobi(a, 2*n + 1), kronecker(a, 2*n + 1))
}
```

Jacobsthal numbers

```
func jacobsthal(n) {
    lucasU(1, -2, n)
}

func lucas_jacobsthal(n) {
    lucasV(1, -2, n)
}

say "First 30 Jacobsthal numbers:"
say 30.of(jacobsthal)

say "\nFirst 30 Jacobsthal-Lucas numbers:"
say 30.of(lucas_jacobsthal)

say "\nFirst 20 Jacobsthal oblong numbers:"
say 21.of(jacobsthal).cons(2, {|a,b| a * b })

say "\nFirst 20 Jacobsthal primes:";
say (1..Inf -> lazy.map(jacobsthal).grep{.is_prime}.first(20))
```

Output:

```
First 30 Jacobsthal numbers:
[0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, 683, 1365, 2731, 5461, 10923, 21845, 43691, 87381, 174763, 34

First 30 Jacobsthal-Lucas numbers:
[2, 1, 5, 7, 17, 31, 65, 127, 257, 511, 1025, 2047, 4097, 8191, 16385, 32767, 65537, 131071, 262145, 52

First 20 Jacobsthal oblong numbers:
[0, 1, 3, 15, 55, 231, 903, 3655, 14535, 58311, 232903, 932295, 3727815, 14913991, 59650503, 238612935,

First 20 Jacobsthal primes:
[3, 5, 11, 43, 683, 2731, 43691, 174763, 2796203, 715827883, 2932031007403, 768614336404564651, 2014876
```

Jaro similarity

```
func jaro(s, t) {

    return 1 if (s == t)

    var s_len = s.len
    var t_len = t.len

    var match_distance = ((s_len `max` t_len) // 2 - 1)

    var s_matches = []
    var t_matches = []

    var matches = 0
    var transpositions = 0

    for i (^s_len) {
        var start = (0 `max` i-match_distance)
        var end = (i+match_distance `min` t_len-1)

        for k (start..end) {
            t_matches[k] && next
            s[i] == t[k] || next
            s_matches[i] = true
            t_matches[k] = true
            matches++
            break
        }
    }

    return 0 if (matches == 0)

    var k = 0
    for i (^s_len) {
        s_matches[i] || next
        while (!t_matches[k]) { ++k }
        s[i] == t[k] || ++transpositions
        ++k
    }

    ((matches / s_len) +
      (matches / t_len) +
        ((matches - transpositions/2) / matches)) / 3
}

for pair in [
    [%c"MARTHA",    %c"MARHTA"],
    [%c"DIXON",     %c"DICKSONX"],
    [%c"JELLYFISH", %c"SMELLYFISH"],
] {
    say "jaro(#{pair.map{.join.dump}.join(', ')}) = #{'%.10f' % jaro(pair...)}"
}
```

Output:

```
jaro("MARTHA", "MARHTA") = 0.9444444444
jaro("DIXON", "DICKSONX") = 0.7666666667
jaro("JELLYFISH", "SMELLYFISH") = 0.8962962963
```

Jensen's Device

```
var i
func sum (i, lo, hi, term) {
    var temp = 0
    for (*i = lo; *i <= hi; (*i)++) {
        temp += term.run
    }
    return temp
}
say sum(\i, 1, 100, { 1 / i })
```

Output:

```
5.187377517639620260805117675658 25
```

Jewels and stones

```
func countJewels(s, j) {
    s.chars.count { |c|
        j.contains(c)
    }
}

say countJewels("aAAbbbb", "aA")    #=> 3
say countJewels("ZZ", "z")          #=> 0
```

JortSort

```
func jort_sort(array) { array == array.sort }
```

Josephus problem

Iterative:

```
func josephus(n, k) {
    var prisoners = @^n
    while (prisoners.len > 1) {
        prisoners.rotate!(k - 1).shift
    }
    return prisoners[0]
}
```

Recursive:

```
func josephus(n, k) {
    n == 1 ? 0 : ((__FUNC__(n-1, k) + k) % n)
}
```

Calling the function:

```
var survivor = josephus(41, 3)
say "Prisoner #{survivor} survived."
```

Output:

```
Prisoner 30 survived.
```

JSON

```
var json = require('JSON').new
var data = json.decode('{"blue": [1, 2], "ocean": "water"}')
say data
data{:ocean} = Hash(water => %w[fishy salty])
say json.encode(data)
```

Output:

```
Hash(
    'blue' => [1, 2],
    'ocean' => 'water'
    )
{"blue":[1,2],"ocean":{"water":["fishy","salty"]}}
```

Julia set

```
require('Imager')

var (w, h) = (640, 480)
var img = %s'Imager'.new(xsize => w, ysize => h, channels => 3)

var maxIter = 50
var c = Complex(-0.388, 0.613)

var color = %s'Imager::Color'.new('#000000')

for x,y in (^w ~X ^h) {
    var i = maxIter
    var z = Complex((x - w/2) / w * 3, (y - h/2) / h * 2)
    while (z.abs < 2 && --i) {
        z = (z*z + c)
    }
    color.set(hsv => [i / maxIter * 360, 1, i])
    img.setpixel(x => x, y => y, color => color)
}

img.write(file => "JuliaSet_sidef.png")
```

This version generates an ASCII representation:

```
var (w, h) = (141, 50)

var maxIter = 40
var c = Complex(-0.8, 0.156)

for y in ^h {
    for x in ^w {
        var i = maxIter
        var z = Complex(3 * (x - w/2) / w, 2 * (y - h/2) / h)
        while (z.abs < 2 && --i) {
            z = (z*z + c)
        }
        print (i > 0 ? ' ' : '#')
    }
    print "\n"
}
```

K-d tree

```
struct Kd_node {
    d,
    split,
    left,
    right,
}

struct Orthotope {
    min,
    max,
}

class Kd_tree(n, bounds) {

    method init {
        n = self.nk2(0, n);
    }

    method nk2(split, e) {
        return(nil) if (e.len <= 0);
        var exset = e.sort_by { _[split] }
        var m = (exset.len // 2);
        var d = exset[m];
        while ((m+1 < exset.len) && (exset[m+1][split] == d[split])) {
            ++m;
        }

        var s2 = ((split + 1) % d.len);     # cycle coordinates
        Kd_node(d: d, split: split,
                left:  self.nk2(s2, exset.first(m)),
                right: self.nk2(s2, exset.last(m-1)));
    }
}

struct T3 {
    nearest,
    dist_sqd = Inf,
    nodes_visited = 0,
}

func find_nearest(k, t, p) {
    func nn(kd, target, hr, max_dist_sqd) {
```

```
        kd || return T3(nearest: [0]*k);

        var nodes_visited = 1;
        var s = kd.split;
        var pivot = kd.d;
        var left_hr = Orthotope(hr.min, hr.max);
        var right_hr = Orthotope(hr.min, hr.max);
        left_hr.max[s] = pivot[s];
        right_hr.min[s] = pivot[s];

        var nearer_kd;
        var further_kd;
        var nearer_hr;
        var further_hr;
        if (target[s] <= pivot[s]) {
            (nearer_kd, nearer_hr) = (kd.left, left_hr);
            (further_kd, further_hr) = (kd.right, right_hr);
        }
        else {
            (nearer_kd, nearer_hr) = (kd.right, right_hr);
            (further_kd, further_hr) = (kd.left, left_hr);
        }

        var n1 = nn(nearer_kd, target, nearer_hr, max_dist_sqd);
        var nearest = n1.nearest;
        var dist_sqd = n1.dist_sqd;
        nodes_visited += n1.nodes_visited;

        if (dist_sqd < max_dist_sqd) {
            max_dist_sqd = dist_sqd;
        }
        var d = (pivot[s] - target[s] -> sqr);
        if (d > max_dist_sqd) {
            return T3(nearest: nearest, dist_sqd: dist_sqd, nodes_visited: nodes_visited);
        }
        d = (pivot ~Z- target »sqr»() «+»);
        if (d < dist_sqd) {
            nearest = pivot;
            dist_sqd = d;
            max_dist_sqd = dist_sqd;
        }

        var n2 = nn(further_kd, target, further_hr, max_dist_sqd);
        nodes_visited += n2.nodes_visited;
        if (n2.dist_sqd < dist_sqd) {
            nearest = n2.nearest;
            dist_sqd = n2.dist_sqd;
        }

        T3(nearest: nearest, dist_sqd: dist_sqd, nodes_visited: nodes_visited);
    }

    return nn(t.n, p, t.bounds, Inf);
}

func show_nearest(k, heading, kd, p) {
    print <<-"END"
        #{heading}:
        Point:            [#{p.join(',')}]
        END
    var n = find_nearest(k, kd, p);
    print <<-"END"
        Nearest neighbor: [#{n.nearest.join(',')}]
        Distance:         #{sqrt(n.dist_sqd)}
        Nodes visited:    #{n.nodes_visited()}
```

```
        END
}

func random_point(k) { k.of { 1.rand } }
func random_points(k, n) { n.of { random_point(k) } }

var kd1 = Kd_tree([[2, 3],[5, 4],[9, 6],[4, 7],[8, 1],[7, 2]],
             Orthotope(min: [0, 0], max: [10, 10]));
show_nearest(2, "Wikipedia example data", kd1, [9, 2]);

var N = 1000
var t0 = Time.micro
var kd2 = Kd_tree(random_points(3, N), Orthotope(min: [0,0,0], max: [1,1,1]))

var t1 = Time.micro
show_nearest(2,
    "k-d tree with #{N} random 3D points (generation time: #{t1 - t0}s)",
     kd2, random_point(3))
```

Output:

```
Wikipedia example data:
Point:            [9,2]
Nearest neighbor: [8,1]
Distance:         1.414213562373095048801688724209698078569671875376 95
Nodes visited:    3

k-d tree with 1000 random 3D points (generation time: 0.25858s):
Point:            [0.289610993894835321409419086325854632457039511850 91,0.533837355701575215481695618 46
Nearest neighbor: [0.283441308978032486364070834747334854422762254907 65,0.542242559449243043821306375 84
Distance:         0.060417033539248701334116598858185733450110763495 73
Nodes visited:    30
```

Kaprekar numbers

```
var kapr = Set()

for n in (1..15) {
    var k = (10**n - 1)
    k.udivisors.each {|d|
        var dp = k/d
        kapr << (dp == 1 ? d : d*invmod(d, dp))
    }
}

say kapr.grep { .<= 1e4 }.sort

for n in (6 .. 14) {
    var k = (10**n - 1)
    printf("Kaprekar numbers <= 10^%2d:  %5d\n", n, kapr.count_by { .<= k })
}
```

Output:

```
[1, 9, 45, 55, 99, 297, 703, 999, 2223, 2728, 4879, 4950, 5050, 5292, 7272, 7777, 9999]
Kaprekar numbers <= 10^ 6:     54
Kaprekar numbers <= 10^ 7:     62
Kaprekar numbers <= 10^ 8:     91
Kaprekar numbers <= 10^ 9:    102
Kaprekar numbers <= 10^10:    132
Kaprekar numbers <= 10^11:    149
Kaprekar numbers <= 10^12:    264
Kaprekar numbers <= 10^13:    281
Kaprekar numbers <= 10^14:    316
```

Keyboard input/Flush the keyboard buffer

```
var k = frequire('Term::ReadKey');

k.ReadMode('restore');    # Flush the keyboard and returns input stream to initial state
# ReadMode 0;             # Numerical equivalent of keyboard restore (move comment marker to use
instead)

# A more complete example for use in keyboard handler programming.
# We should also check we are being used in an interactive context (not done here).

k.ReadMode('cbreak');

# Flush the keyboard in terminal character break mode
while (k.ReadKey(-1) != nil) {
   # Do nothing
}

# Don't forget to restore the readmode, when we are finished using the keyboard
k.ReadMode('restore');
```

Keyboard input/Obtain a Y or N response

```
func prompt_yn {
    static rk = frequire('Term::ReadKey')
    rk.ReadMode(4)     # change to raw input mode

    var key = ''
    while (key !~ /[yn]/i) {
        while (rk.ReadKey(-1) != nil) {}   # discard any previous input
        print "Type Y/N: "
        say (key = rk.ReadKey(0))          # read a single character
    }

    rk.ReadMode(0)     # reset the terminal to normal mode
    return key.uc
}

var key = prompt_yn()
say "You typed: #{key}"
```

Output:

```
Type Y/N: a
Type Y/N: b
Type Y/N: c
Type Y/N: y
You typed: Y
```

Knapsack problem/0-1

```
var raw = <<'TABLE'
map,                       9, 150
compass,                  13,  35
water,                   153, 200
sandwich,                 50, 160
glucose,                  15,  60
tin,                      68,  45
banana,                   27,  60
apple,                    39,  40
cheese,                   23,  30
beer,                     52,  10
suntancream,              11,  70
camera,                   32,  30
T-shirt,                  24,  15
trousers,                 48,  10
umbrella,                 73,  40
waterproof trousers,      42,  70
waterproof overclothes,   43,  75
note-case,                22,  80
sunglasses,                7,  20
towel,                    18,  12
socks,                     4,  50
book,                     30,  10
TABLE

struct KnapsackItem {
    String name,
    Number weight,
    Number value,
}

var items = []
raw.each_line{ |row|
    var fields = row.split(/\s*,\s*/)
    items << KnapsackItem(
          name: fields[0],
        weight: fields[1].to_n,
         value: fields[2].to_n,
    )
}

var max_weight = 400
var p = [
    items.len.of { [[0, []], max_weight.of(nil)...] }...,
    max_weight.inc.of {[0, []]}
]

func optimal(i, w) {
    if (!defined p[i][w]) {
        var item = items[i];
        if (item.weight > w) {
            p[i][w] = optimal(i.dec, w)
        }
```

```
        else {
            var x = optimal(i.dec, w)
            var y = optimal(i.dec, w - item.weight)

            if (x[0] > (y[0] + item.value)) {
                p[i][w] = x;
            }
            else {
                p[i][w] = [y[0] + item.value, [y[1]..., item.name]]
            }
        }
    }
    return p[i][w]
}

var sol = optimal(items.end, max_weight)
say "#{sol[0]}: #{sol[1].join(' ')}"
```

Output:

```
1030: map compass water sandwich glucose banana suntancream waterproof trousers waterproof overclothes
```

Knapsack problem/Bounded

```
var raw = <<'TABLE'
map           9     150      1
compass      13      35      1
water       153     200      2
sandwich     50      60      2
glucose      15      60      2
tin          68      45      3
banana       27      60      3
apple        39      40      3
cheese       23      30      1
beer         52      10      1
suntancream  11      70      1
camera       32      30      1
T-shirt      24      15      2
trousers     48      10      2
umbrella     73      40      1
w_trousers   42      70      1
w_overcoat   43      75      1
note-case    22      80      1
sunglasses    7      20      1
towel        18      12      2
socks         4      50      1
book         30      10      2
TABLE

struct KnapsackItem {
    String name,
    Number weight,
    Number value,
    Number quant,
}

var items = []
raw.each_line{ |row|
    var fields = row.words;
```

```
    items << KnapsackItem(
          name: fields[0],
        weight: fields[1].to_n,
         value: fields[2].to_n,
         quant: fields[3].to_n,
    )
}

func pick(weight, pos) is cached {

    if (pos.is_neg || weight.is_neg || weight.is_zero) {
        return (0, 0, [])
    }

    var (bv=0, bi=0, bw=0, bp=[])
    var item = items[pos];

    for i in range(0, item.quant) {
        break if (i*item.weight > weight)
        var (v, w, p) = pick(weight - i*item.weight, pos.dec)
        next if ((v += i*item.value) <= bv)
        (bv, bi, bw, bp) = (v, i, w, p)
    }

    (bv, bw + bi*item.weight, [bp..., bi])
}

var (v, w, p) = pick(400, items.end)
p.range.each { |i|
    say "#{p[i]} of #{items[i].name}" if p[i].is_pos
}
say "Value: #{v}; Weight: #{w}"
```

Output:

```
1 of map
1 of compass
1 of water
2 of glucose
3 of banana
1 of cheese
1 of suntancream
1 of w_overcoat
1 of note-case
1 of sunglasses
1 of socks
Value: 1010; Weight: 396
```

Knapsack problem/Continuous

```
var items =
[
        [:beef,    3.8, 36],
        [:pork,    5.4, 43],
        [:ham,     3.6, 90],
        [:greaves, 2.4, 45],
        [:flitch,  4.0, 30],
        [:brawn,   2.5, 56],
        [:welt,    3.7, 67],
        [:salami,  3.0, 95],
        [:sausage, 5.9, 98],
].sort {|a,b| b[2]/b[1] <=> a[2]/a[1] };

var (limit, value) = (15, 0);
print "Item   Fraction Weight Value\n";

items.each { |item|
        var ratio = (item[1] > limit ? limit/item[1] : 1);
        value += item[2]*ratio;
        limit -= item[1];
        if (ratio == 1) {
            printf("%-8s %4s %7.2f %6.2f\n", item[0], 'all', item[1], item[2]);
        }
        else {
            printf("%-8s %-4.2f %7.2f %6.2f\n", item[0], ratio, item[1]*ratio, item[2]*ratio);
            break;
        }
};

say "#{'-'*28}\ntotal value: #{'%.14g' % value }";
```

Output:

```
Item   Fraction Weight Value
salami    all    3.00  95.00
ham       all    3.60  90.00
brawn     all    2.50  56.00
greaves   all    2.40  45.00
welt     0.95    3.50  63.38
----------------------------
total value: 349.37837837838
```

Knapsack problem/Unbounded

```
struct KnapsackItem {
    Number volume,
    Number weight,
    Number value,
    String name,
}

var items = [
    KnapsackItem(25,  3, 3000, "panacea")
    KnapsackItem(15,  2, 1800, "ichor"  )
    KnapsackItem( 2, 20, 2500, "gold"   )
]

var (
    max_weight = 250,
```

```
    max_vol = 250,
    vsc = 1000,
    wsc = 10
)

func solve(i, w, v) is cached {
    return [0, []] if i.is_neg;

    var x = solve(i.dec, w, v);

    var (w1, v1);
    Inf.times { |t|
        var item = items[i];
        break if ((w1 = (w - t*item.weight)).is_neg)
        break if ((v1 = (v - t*item.volume)).is_neg)

        var y = solve(i.dec, w1, v1);
        if ((var tmp = (y[0] + t*item.value)) > x[0]) {
            x = [tmp, [y[1]..., [i, t]]];
        }
    }

    return x
}

var x = solve(items.end, max_weight, max_vol)

print <<"EOT"
Max value #{x[0]}, with:
    Item        Qty     Weight   Vol    Value
#{"-" * 50}
EOT

var (wtot=0, vtot=0);
x[1].each { |s|
    var item = items[s[0]];
    "    #{item.name}:\t% 3d  % 8d% 8g% 8d\n".printf(
        s[1],
        item.weight * s[1] / wsc,
        item.volume * s[1] / vsc,
        item.value  * s[1]
    );
    wtot += (item.weight * s[1]);
    vtot += (item.volume * s[1]);
}

print <<"EOT"
#{"-" * 50}
    Total:\t     #{"%8d%8g%8d" % (wtot/wsc, vtot/vsc, x[0])}
EOT
```

Output:

```
Max value 54500, with:
    Item        Qty     Weight   Vol    Value
--------------------------------------------------
    panacea:      9         2   0.225   27000
    gold:        11        22   0.022   27500
--------------------------------------------------
    Total:                 24   0.247   54500
```

Knight's tour

```
var board = []
var I = 8
var J = 8
var F = (I*J > 99 ? '%3d' : '%2d')

var (i, j) = (I.irand, J.irand)

func from_algebraic(square) {
     if (var match = square.match(/^([a-z])([0-9])\z/)) {
         return(I - Num(match[1]), match[0].ord - 'a'.ord)
     }
     die "Invalid block square: #{square}"
}

func possible_moves(i,j) {
    gather {
        for ni,nj in [
            [i-2,j-1], [i-2,j+1], [i-1,j-2], [i-1,j+2],
            [i+1,j-2], [i+1,j+2], [i+2,j-1], [i+2,j+1],
        ] {
            if ((ni ~~ ^I) && (nj ~~ ^J) && !board[ni][nj]) {
                take([ni, nj])
            }
        }
    }
}

func to_algebraic(i,j) {
    ('a'.ord + j).chr + Str(I - i)
}

if (ARGV[0]) {
    (i, j) = from_algebraic(ARGV[0])
}

var moves = []
for move in (1 .. I*J) {
    moves << to_algebraic(i, j)
    board[i][j] = move
    var min = [9]
    for target in possible_moves(i, j) {
        var (ni, nj) = target...
        var nxt = possible_moves(ni, nj).len
        if (nxt < min[0]) {
            min = [nxt, ni, nj]
        }
    }

    (i, j) = min[1,2]
}

say (moves/4 -> map { .join(', ') }.join("\n") + "\n")

for i in ^I {
    for j in ^J {
        (i%2 == j%2) && print "\e[7m"
        F.printf(board[i][j])
        print "\e[0m"
    }
    print "\n"
}
```

Knuth's algorithm S

```
func s_of_n_creator(n) {
    var i = 0
    var sample = []
    { |item|
        if (++i <= n) {
            sample << item;
        }
        elsif (i.rand < n) {
            sample[n.rand] = item;
        }
        sample;
    }
}

var items = 0..9;
var bin = [];

100000.times {
    var s_of_n = s_of_n_creator(3);
    var sample = []
    for item in items {
        sample = s_of_n(item);
    }
    for s in sample {
        bin[s] := 0 ++;
    }
}

say bin;
```

Output:

```
[30056, 29906, 30058, 29986, 30062, 29748, 29989, 29985, 30126, 30084]
```

Knuth's power tree

```
var lvl = [[1]]
var p = Hash(1 => 0)

func path(n) is cached {
    n || return []
    while (n !~ p) {
        var q = []
        for x in lvl[0] {
            for y in path(x) {
                break if (x+y ~~ p)
                y = x+y
                p{y} = x
                q << y
            }
        }
        lvl[0] = q
    }
    path(p{n}) + [n]
}

func tree_pow(x, n) {
    var r = Hash(0 => 1, 1 => x)
    var p = 0
    for i in path(n) {
        r{i} = (r{i-p} * r{p})
        p = i
    }
    r{n}
}

func show_pow(x, n) {
    var fmt = ("%d: %s\n" + ["%g^%s = %f", "%s^%s = %s"][x.is_int] + "\n")
    print(fmt % (n, path(n), x, n, tree_pow(x, n)))
}

for x in ^18 { show_pow(2, x) }
show_pow(1.1, 81)
show_pow(3, 191)
```

Knuth shuffle

```
func knuth_shuffle(a) {
    for i (a.len ^.. 1) {
        var j = i.irand
        a[i, j] = a[j, i]
    }
    return a
}

say knuth_shuffle(@(1..10))
```

Output:

```
[5, 8, 4, 7, 10, 2, 9, 3, 1, 6]
```

Koch curve

Using the LSystem class defined at Hilbert curve.

```
var rules = Hash(
    F => 'F+F--F+F',
)

var lsys = LSystem(
    width:  800,
    height: 800,

    xoff: -210,
    yoff: -90,

    len:   8,
    angle: 60,
    color: 'dark green',
)

lsys.execute('F--F--F', 4, "koch_snowflake.png", rules)
```

Output image

Kolakoski sequence

```
func create_generator(arr) {
    Enumerator({|f|
        var s = []
        var i = 0
        loop {
            var t = arr[i++ % arr.len]
            s << t
            f(var v = s.shift)
            s << (v-1).of(t)...
        }
    })
}

var tests = [
    [20, [1,2]],
    [20, [2,1]],
    [30, [1,3,1,2]],
    [30, [1,3,2,1]]
]

for num,arr in (tests) {
    say "\nFirst #{num} of the sequence generated by #{arr}:"
    var res = create_generator(arr).first(num)
    var rle = res.run_length.map{.tail}
    say "#{res}\nPossible Kolakoski sequence? #{res.first(rle.len) == rle}"
}
```

Output:

```
First 20 of the sequence generated by [1, 2]:
[1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1]
Possible Kolakoski sequence? true

First 20 of the sequence generated by [2, 1]:
[2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2]
Possible Kolakoski sequence? true

First 30 of the sequence generated by [1, 3, 1, 2]:
[1, 3, 3, 3, 1, 1, 1, 2, 2, 2, 1, 3, 1, 2, 2, 1, 1, 3, 3, 1, 2, 2, 2, 1, 3, 3, 1, 1, 2, 1]
Possible Kolakoski sequence? true

First 30 of the sequence generated by [1, 3, 2, 1]:
[1, 3, 3, 3, 2, 2, 2, 1, 1, 1, 1, 1, 3, 3, 2, 2, 1, 1, 3, 2, 1, 1, 1, 1, 3, 3, 3, 2, 2, 1]
Possible Kolakoski sequence? false
```

Kosaraju

```
func korasaju(Array g) {
    # 1. For each vertex u of the graph, mark u as unvisited. Let L be empty.
    var vis = g.len.of(false)
    var L   = []
    var x   = g.end
    var t   = g.len.of { [] }

    # Recursive
    func visit(u) {
        if (!vis[u]) {
            vis[u] = true
            g[u].each {|v|
                visit(v)
                t[v] << u
            }
            L[x--] = u
        }
    }

    # 2. For each vertex u of the graph do visit(u)
    g.range.each {|u|
        visit(u)
    }

    var c = []

    # 3. Recursive subroutine:
    func assign(u, root) {
        if (vis[u]) {
            vis[u] = false
            c[u] = root
            t[u].each {|v|
                assign(v, root)
            }
        }
    }

    # 3. For each element u of L in order, do assign(u, u)
    L.each {|u|
        assign(u, u)
    }

    return c
}

var g = [[1], [2], [0], [1, 2, 4], [3, 5], [2, 6], [5], [4, 6, 7]]
say korasaju(g)
```

Output:

```
[0, 0, 0, 3, 3, 5, 5, 7]
```

Kronecker product

```
func kronecker_product(a, b) {
    a ~X b -> map { _[0] ~X* _[1] }
}

kronecker_product([[1, 2], [3, 4]],
                  [[0, 5], [6, 7]]).each { .say }

say ''
kronecker_product([[0,1,0],  [1,1,1],   [0,1,0]],
                  [[1,1,1,1],[1,0,0,1], [1,1,1,1]]).each { .say }
```

Output:

```
[0, 5, 0, 10]
[6, 7, 12, 14]
[0, 15, 0, 20]
[18, 21, 24, 28]

[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]
```

Kronecker product based fractals

```
func kronecker_product (a, b) { a ~X b -> map { _[0] ~X* _[1] } }

func kronecker_fractal(pattern, order=4) {
    var kronecker = pattern
    { kronecker = kronecker_product(kronecker, pattern) } * order
    return kronecker
}

var vicsek = [[0,1,0], [1,1,1], [0,1,0]]
var carpet = [[1,1,1], [1,0,1], [1,1,1]]
var six    = [[0,1,1,1,0], [1,0,0,0,1], [1,0,0,0,0],
              [1,1,1,1,0], [1,0,0,0,1], [1,0,0,0,1], [0,1,1,1,0]]

require("Imager")

for name,shape,order in [
    [:vicsek, vicsek, 4],
    [:carpet, carpet, 4],
    [:six,    six,    3],
] {
    var pat = kronecker_fractal(shape, order)
    var img = %O<Imager>.new(xsize => pat[0].len, ysize => pat.len)
    for x,y in (^pat[0].len ~X ^pat.len) {
        img.setpixel(x => x, y => y, color => (pat[y][x] ? <255 255 32> : <16 16 16>))
    }
    img.write(file => "kronecker-#{name}-sidef.png")
}
```

Output images:

```
* [Kronecker six](https://github.com/trizen/rc/blob/master/img/kronecker-six-sidef.png)
* [Kronecker carpet](https://github.com/trizen/rc/blob/master/img/kronecker-carpet-sidef.png)
* [Kronecker vicsek](https://github.com/trizen/rc/blob/master/img/kronecker-vicsek-sidef.png)
```

Lah numbers

```
func lah(n, k) {
    stirling3(n, k)
    #binomial(n-1, k-1) * n!/k!     # alternative formula
}

const r = (0..12)

var triangle = r.map {|n| 0..n -> map {|k| lah(n, k) } }
var widths   = r.map {|n| r.map {|k| (triangle[k][n] \\ 0).len }.max }

say ('n\k ', r.map {|n| "%*s" % (widths[n], n) }.join(' '))

r.each {|n|
    var str = ('%-3s ' % n)
    str += triangle[n].map_kv {|k,v| "%*s" % (widths[k], v) }.join(' ')
    say str
}

with (100) {|n|
    say "\nMaximum value from the L(#{n}, *) row:"
    say { lah(n, _) }.map(^n).max
}
```

Output:

```
n\k 0         1          2          3          4          5         6        7       8      9   10  11
0   1
1   0         1
2   0         2          1
3   0         6          6          1
4   0        24         36         12          1
5   0       120        240        120         20          1
6   0       720       1800       1200        300         30         1
7   0      5040      15120      12600       4200        630        42        1
8   0     40320     141120     141120      58800      11760      1176       56       1
9   0    362880    1451520    1693440     846720     211680     28224     2016      72      1
10  0   3628800   16329600   21772800   12700800    3810240    635040    60480    3240     90    1
11  0  39916800  199584000  299376000  199584000   69854400  13970880  1663200  118800   4950  110   1
12  0 479001600 2634508800 4390848000 3293136000 1317254400 307359360 43908480 3920400 217800 7260 132

Maximum value from the L(100, *) row:
44519005448993144810881324947684737529186447692709328597242209638906324913313742508392928375354932241400
```

Langton's ant

```
define dirs = [[1,0], [0,-1], [-1,0], [0,1]]
define size = 100

enum |White, Black|
var plane = size.of { size.of (White) }

var (x, y) = ([size >> 1] * 2)...
var dir = dirs.len.irand

var moves = 0
loop {
    (x >= 0) && (y >= 0) && (x < size) && (y < size) || break

    given (plane[x][y]) {
        when (White) { dir--; plane[x][y] = Black }
        when (Black) { dir++; plane[x][y] = White }
    }

    ++moves
    [[\x, \y], dirs[dir %= dirs.len]].zip {|a,b| *a += b }
}

say "Out of bounds after #{moves} moves at (#{x}, #{y})"
plane.map{.map {|square| square == Black ? '#' : '.' }}.each{.join.say}
```

Largest difference between adjacent primes

```
func prime_gap_records(upto) {

    var gaps = []
    var p = 2

    each_prime(p.next_prime, upto, {|q|
        gaps[q-p] := p
        p = q
    })

    gaps.grep { defined(_) }
}

var upto   = 1e8
var primes = prime_gap_records(upto)

for n in (2 .. upto.ilog10) {

    var b = primes.last_by {|p| p < 10**n } \\ break

    printf("Largest prime gap up to 10^%s is %3s between %s and %s\n",
        n, b.next_prime - b, b, b.next_prime)
}
```

Output:

```
Largest prime gap up to 10^2 is   8 between 89 and 97
Largest prime gap up to 10^3 is  20 between 887 and 907
Largest prime gap up to 10^4 is  36 between 9551 and 9587
Largest prime gap up to 10^5 is  72 between 31397 and 31469
Largest prime gap up to 10^6 is 114 between 492113 and 492227
Largest prime gap up to 10^7 is 154 between 4652353 and 4652507
Largest prime gap up to 10^8 is 220 between 47326693 and 47326913
```

Largest five adjacent number

```
var k = 5
var n = 1e1000.irand

say "length(n) = #{n.len}"

var c = n.digits.cons(k)

say ("Min #{k}-digit sub-number: ", c.min_by { .digits2num }.flip.join)
say ("Max #{k}-digit sub-number: ", c.max_by { .digits2num }.flip.join)
```

Output:

```
length(n) = 1000
Min 5-digit sub-number: 00072
Max 5-digit sub-number: 99861
```

Largest int from concatenated ints

```
func maxnum(nums) {
    nums.sort {|x,y|  "#{y}#{x}" <=> "#{x}#{y}" };
}

[[54, 546, 548, 60], [1, 34, 3, 98, 9, 76, 45, 4]].each { |c|
    say maxnum(c).join.to_num;
}
```

Output:

```
6054854654
998764543431
```

Largest number divisible by its digits

```
func largest_number(base) {

    var digits = @(base ^.. 1)

    digits.each {|k|
        digits.variations(k, {|*a|
            var n = Number(a.join, base)
            if (a.all {|d| d.divides(n) }) {
                return n
            }
        })
    }
}

say largest_number(10)   #=> 9867312
```

Largest palindrome product

```
func largest_palindrome_product (n) {

    for k in ((10**n - 1) `downto` 10**(n-1)) {
        var t = Num("#{k}#{Str(k).flip}")

        t.divisors.each {|d|
            if ((d.len == n) && ((t/d).len == n)) {
                return (d, t/d)
            }
        }
    }
}

for n in (2..9) {
    var (a,b) = largest_palindrome_product(n)
    say "Largest palindromic product of two #{n}-digit integers: #{a} * #{b} = #{a*b}"
}
```

Output:

```
Largest palindromic product of two 2-digit integers: 91 * 99 = 9009
Largest palindromic product of two 3-digit integers: 913 * 993 = 906609
Largest palindromic product of two 4-digit integers: 9901 * 9999 = 99000099
Largest palindromic product of two 5-digit integers: 99681 * 99979 = 9966006699
Largest palindromic product of two 6-digit integers: 999001 * 999999 = 999000000999
Largest palindromic product of two 7-digit integers: 9997647 * 9998017 = 99956644665999
Largest palindromic product of two 8-digit integers: 99990001 * 99999999 = 9999000000009999
Largest palindromic product of two 9-digit integers: 999920317 * 999980347 = 999900665566009999
```

Largest prime factor

```
var n = 600851475143

say gpf(n)             #=> 6857
say factor(n).last     #=> 6857
```

Largest product in a grid

```
var text = <<'EOT'
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
EOT

func horizontal(N, i, j, matrix) {
    N.of {|k| matrix[i][j+k] }
}

func diagonal(N, i, j, matrix) {
    N.of {|k| matrix[i+k][j+k] }
}

var matrix = Matrix(text.lines.map{ .nums }...)

var reversed_matrix   = matrix.horizontal_flip
var transposed_matrix = matrix.transpose

define (
    CHECK_DIAGONALS = false   # true to also check diagonals
)

const e = matrix.end

for N in (1..6) {

    var products = gather {
        for i in (0..e), j in (0..e) {

            (j+N < e) || next

            # Horizontal and vertical
            take(horizontal(N, i, j, matrix))
            take(horizontal(N, i, j, transposed_matrix))

            CHECK_DIAGONALS || next
            (i+N < e)       || next

            # Left-to-right and right-to-left diagonals
            take(diagonal(N, i, j, matrix))
            take(diagonal(N, i, j, reversed_matrix))
        }
    }
```

```
    var nums = products.max_by { .prod }
    say "Largest product of #{N} adjacent elements: prod(#{nums}) = #{nums.prod}"
}
```

Output:

```
Largest product of 1 adjacent elements: prod([99]) = 99
Largest product of 2 adjacent elements: prod([95, 97]) = 9215
Largest product of 3 adjacent elements: prod([91, 88, 97]) = 776776
Largest product of 4 adjacent elements: prod([66, 91, 88, 97]) = 51267216
Largest product of 5 adjacent elements: prod([62, 99, 69, 82, 67]) = 2326829868
Largest product of 6 adjacent elements: prod([99, 69, 82, 67, 59, 85]) = 188210512710
```

Last Friday of each month

```
require('DateTime')
var (year=2016) = ARGV.map{.to_i}...

for month (1..12) {
   var dt = %O<DateTime>.last_day_of_month(year => year, month => month)
   while (dt.day_of_week != 5) {
      dt.subtract(days => 1)
   }
   say dt.ymd
}
```

Output:

```
$ sidef lastfriday.sf 2012
2012-01-27
2012-02-24
2012-03-30
2012-04-27
2012-05-25
2012-06-29
2012-07-27
2012-08-31
2012-09-28
2012-10-26
2012-11-30
2012-12-28
```

Last letter-first letter

```
class LLFL(Array words) {

    has f = Hash()

    method init {
        words.each {|w|
            var m = w.match(/^(.).*(.)$/)
            f{m[0]}{w} = m[1]
        }
    }

    method longest_chain {

        var stack   = []
        var longest = []

        func poke(c) {
            var h = f{c}

            h.each_k { |word|
                var v = h.delete(word)
                stack.push(word)
                longest[] = stack[] if (stack.len > longest.len)
                __FUNC__(v) if f.exists(v)
                stack.pop
                h{word} = v
            }
        }

        f.each_k { poke(_) }
        return longest
    }
}

var words = %w(
  audino bagon baltoy banette bidoof braviary bronzor carracosta charmeleon
  cresselia croagunk darmanitan deino emboar emolga exeggcute gabite
  girafarig gulpin haxorus heatmor heatran ivysaur jellicent jumpluff kangaskhan
  kricketune landorus ledyba loudred lumineon lunatone machamp magnezone mamoswine
  nosepass petilil pidgeotto pikachu pinsir poliwrath poochyena porygon2
  porygonz registeel relicanth remoraid rufflet sableye scolipede scrafty seaking
  sealeo silcoon simisear snivy snorlax spoink starly tirtouga trapinch treecko
  tyrogue vigoroth vulpix wailord wartortle whismur wingull yamask
)

var obj = LLFL(words)
var longest = obj.longest_chain()

say "#{longest.len}: #{longest.join(' ')}"
```

Output:

```
23: machamp petilil landorus scrafty yamask kricketune exeggcute emboar registeel loudred darmanitan no
```

Last list item

With sorting

```
var k = 2
var list = [6, 81, 243, 14, 25, 49, 123, 69, 11]

while (list.len >= k) {
    var a = list.sort!.shift(k)
    list << a.sum
    say "#{a.map{'%3s' % _}.join(' ')} : #{list}"
}
```

Output:

```
  6  11 : [14, 25, 49, 69, 81, 123, 243, 17]
 14  17 : [25, 49, 69, 81, 123, 243, 31]
 25  31 : [49, 69, 81, 123, 243, 56]
 49  56 : [69, 81, 123, 243, 105]
 69  81 : [105, 123, 243, 150]
105 123 : [150, 243, 228]
150 228 : [243, 378]
243 378 : [621]
```

Without sorting

```
var k = 2
var list = [6, 81, 243, 14, 25, 49, 123, 69, 11]

while (list.len >= k) {

    var a = gather {
        k.times {
            take(var v = list.min)
            list.delete_first(v)
        }
    }

    list << a.sum
    say "#{a.map{'%3s' % _}.join(' ')} : #{list}"
}
```

Output:

```
  6  11 : [81, 243, 14, 25, 49, 123, 69, 17]
 14  17 : [81, 243, 25, 49, 123, 69, 31]
 25  31 : [81, 243, 49, 123, 69, 56]
 49  56 : [81, 243, 123, 69, 105]
 69  81 : [243, 123, 105, 150]
105 123 : [243, 150, 228]
150 228 : [243, 378]
243 378 : [621]
```

Leap year

```
func isleap(year) {
    if (year %% 100) {
        return (year %% 400)
    }
    return (year %% 4)
}
```

or a little bit simpler:

```
func isleap(year) { year %% 100 ? (year %% 400) : (year %% 4) }
```

Least common multiple

Built-in:

```
say Math.lcm(1001, 221)
```

Using GCD:

```
func gcd(a, b) {
    while (a) { (a, b) = (b % a, a) }
    return b
}

func lcm(a, b) {
    (a && b) ? (a / gcd(a, b) * b) : 0
}

say lcm(1001, 221)
```

Output:

```
17017
```

Left factorials

Built-in:

```
say 20.of { .left_factorial }
```

Straightforward:

```
func left_factorial(n) {
    ^n -> sum { _! }
}
```

Memory efficient with `Range.reduce()`:

```
func left_factorial(n) {
    ^n -> reduce({ |a,b| a + b! }, 0)
}
```

A faster approach:

```
func left_factorial(n) {
    static cached    = 0
    static factorial = 1
    static leftfact  = 0

    if (n < cached) {
        cached    = 0
        factorial = 1
        leftfact  = 0
    }

    while (n > cached) {
        leftfact  += factorial
        factorial *= ++cached
    }

    leftfact
}
```

Completing the task:

```
for n in (0..10, 20..110 `by` 10) {
    printf("!%d  = %s\n", n, left_factorial(n))
}

for n in (1000..10000 `by` 1000) {
    printf("!%d has %d digits.\n", n, left_factorial(n).len)
}
```

Output:

```
!0  = 0
!1  = 1
!2  = 2
!3  = 4
!4  = 10
!5  = 34
!6  = 154
!7  = 874
!8  = 5914
!9  = 46234
!10  = 409114
!20  = 128425485935180314
!30  = 91579586579510755733953009403l4
!40  = 20935051082417771847631371547939998232420940314
!50  = 62096002783282161263942480669455110881272052560616092042094031l
!60  = 14107493072666957100053082208700052221165624211643994900098037874612892042094031l
!70  = 17363951180298752669971716240928287606555651984960315785085303464481511122159950921652892042094O
!80  = 90608958798769534653451680465029063769402483001195636518432767461975209428969631488200853199184O
!90  = 16695570072624210767034167688394623360733515163575864136345910335924039962404869510225723072235l
!100  = 94278623976582657916059526820683938135475434960105097434539541040707823024959041445883011744261
!110  = 14572298106158529700470672800190607194863519923486072098865804253617928132861554193608329616347
!1000 has 2565 digits.
!2000 has 5733 digits.
!3000 has 9128 digits.
!4000 has 12670 digits.
!5000 has 16322 digits.
!6000 has 20062 digits.
!7000 has 23875 digits.
!8000 has 27749 digits.
!9000 has 31678 digits.
!10000 has 35656 digits.
```

Legendre prime counting function

```
func legendre_phi(x, a) is cached {
    return x if (a <= 0)
    __FUNC__(x, a-1) - __FUNC__(idiv(x, a.prime), a-1)
}

func legendre_prime_count(n) is cached {
    return 0 if (n < 2)
    var a = __FUNC__(n.isqrt)
    legendre_phi(n, a) + a - 1
}

print("e             n   Legendre    builtin\n",
      "-             -   --------    -------\n")

for n in (1 .. 9) {
    printf("%d  %12d %10d %10d\n", n, 10**n,
        legendre_prime_count(10**n), prime_count(10**n))
    assert_eq(legendre_prime_count(10**n), prime_count(10**n))
}
```

Output:

```
e             n   Legendre    builtin
-             -   --------    -------
1            10          4          4
2           100         25         25
3          1000        168        168
4         10000       1229       1229
5        100000       9592       9592
6       1000000      78498      78498
7      10000000     664579     664579
```

Alternative implementation of the Legendre phi function, by counting k-rough numbers <= n.

```
func rough_count (n,k) {

    # Count of k-rough numbers <= n.

    func (n,p) is cached {

        if (p > n.isqrt) {
            return 1
        }

        if (p == 2) {
            return (n >> 1)
        }

        if (p == 3) {
            var t = idiv(n,3)
            return (t - (t >> 1))
        }

        var u = 0
        var t = idiv(n,p)

        for (var q = 2; q < p; q.next_prime!) {

            var v = __FUNC__(t - (t % q), q)

            if (v == 1) {
                u += prime_count(q, p-1)
                break
            }

            u += v
        }

        t - u
    }(n*k, k)
}

func legendre_phi(n, a) {
     rough_count(n, prime(a+1))
}

func legendre_prime_count(n) is cached {
    return 0 if (n < 2)
    var a = __FUNC__(n.isqrt)
    legendre_phi(n, a) + a - 1
}

print("e             n   Legendre    builtin\n",
      "-             -   --------    -------\n")

for n in (1 .. 9) {
    printf("%d  %12d %10d %10d\n", n, 10**n,
        legendre_prime_count(10**n), prime_count(10**n))
    assert_eq(legendre_prime_count(10**n), prime_count(10**n))
}
```

Leonardo numbers

```
func 𝑳(n, 𝑳0 = 1, 𝑳1 = 1, 𝑳add = 1) {
    { (𝑳0, 𝑳1) = (𝑳1, 𝑳0 + 𝑳1 + 𝑳add) } * n
    return 𝑳0
}

say "The first 25 Leonardo numbers:"
say 25.of { 𝑳(_) }

say "\nThe first 25 numbers using 𝑳0 of 0, 𝑳1 of 1, and adder of 0:"
say 25.of { 𝑳(_, 0, 1, 0) }
```

Output:

```
The first 25 Leonardo numbers:
[1, 1, 3, 5, 9, 15, 25, 41, 67, 109, 177, 287, 465, 753, 1219, 1973, 3193, 5167, 8361, 13529, 21891, 35

The first 25 numbers using 𝑳0 of 0, 𝑳1 of 1, and adder of 0:
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711
```

Letter frequency

```
func letter_frequency(File file) {
    file.read.chars.grep{.match(/[[:alpha:]]/)} \
        .group_by {|letter| letter.downcase}    \
        .map_val  {|_, val| val.len}            \
        .sort_by  {|_, val| -val}
}

var top = letter_frequency(File(__FILE__))
top.each{|pair| say "#{pair[0]}: #{pair[1]}"}
```

Output:

```
e: 22
a: 18
l: 16
r: 15
t: 12
p: 9
f: 8
i: 8
c: 7
v: 6
y: 5
n: 5
o: 5
u: 4
h: 4
s: 4
b: 2
g: 2
m: 2
d: 2
q: 2
w: 1
```

Levenshtein distance

Recursive:

```
func lev(s, t) is cached {

    s || return t.len
    t || return s.len

    var s1 = s.ft(1)
    var t1 = t.ft(1)

    s[0] == t[0] ? __FUNC__(s1, t1)
                 : 1+Math.min(
                       __FUNC__(s1, t1),
                       __FUNC__(s,  t1),
                       __FUNC__(s1, t )
                   )
}
```

Iterative:

```
func lev(s, t) {
    var d = [@(0 .. t.len), s.len.of {[_]}...]
    for i,j in (^s ~X ^t) {
        d[i+1][j+1] = (
            s[i] == t[j]
                ? d[i][j]
                : 1+Math.min(d[i][j+1], d[i+1][j], d[i][j])
        )
    }
    d[-1][-1]
}
```

Calling the function:

```
say lev(%c'kitten', %c'sitting');               # prints: 3
say lev(%c'rosettacode', %c'raisethysword');    # prints: 8
```

Levenshtein distance/Alignment

```
func align(s, t) {
    s.chars!.prepend!('^')
    t.chars!.prepend!('^')

    var A = []
    {|i| A[i][0]{@|<d s t>} = (i, s.ft(1, i).join, '~' * i) } << ^s
    {|i| A[0][i]{@|<d s t>} = (i, '-' * i, t.ft(1, i).join) } << ^t

    for i (1 .. s.end) {
      for j (1 .. t.end) {
        if (s[i] != t[j]) {
          A[i][j]{:d} = 1+(
            var min = Math.min(A[i-1][j]{:d}, A[i][j-1]{:d}, A[i-1][j-1]{:d})
          )
          A[i][j]{@|<s t>} = (A[i-1][j]{:d} == min
              ? [A[i-1][j]{:s}+s[i], A[i-1][j]{:t}+'-']
              : (A[i][j-1]{:d} == min
              ? [A[i][j-1]{:s}+'-', A[i][j-1]{:t}+t[j]]
              : [A[i-1][j-1]{:s}+s[i], A[i-1][j-1]{:t}+t[j]]))...
        }
        else {
          A[i][j]{@|<d s t>} = (
              A[i-1][j-1]{:d},
              A[i-1][j-1]{:s}+s[i],
              A[i-1][j-1]{:t}+t[j],
          )
        }
      }
    }
    return [A[-1][-1]{@|<s t>}]
}

align("rosettacode", "raisethysword").each { .say }
```

Output:

```
ro-settac-o-de
raisethysword-
```

Linear congruential generator

```
module LCG {

  # Creates a linear congruential generator and remembers the initial seed.
  class Common(r) {
     has seed = r
  }

  # LCG::Berkeley generates 31-bit integers using the same formula
  # as BSD rand().
  class Berkeley < Common {
    method rand {
      self.r = ((1103515245 * self.r + 12345) & 0x7fff_ffff);
    }
  }

  # LCG::Microsoft generates 15-bit integers using the same formula
  # as rand() from the Microsoft C Runtime.
  class Microsoft < Common {
    method rand {
      self.r = ((214013 * self.r + 2531011) & 0x7fff_ffff);
      self.r >> 16;
    }
  }
}

var lcg1 = LCG::Berkeley(1)
say 5.of { lcg1.rand }

var lcg2 = LCG::Microsoft(1)
say 5.of { lcg2.rand }
```

Output:

```
[1103527590, 377401575, 662824084, 1147902781, 2035015474]
[41, 18467, 6334, 26500, 19169]
```

List comprehensions

```
var n = 20
say gather {
    for x in (1 .. n) {
        for y in (x .. n) {
           for z in (y .. n) {
             take([x,y,z]) if (x*x + y*y == z*z)
           }
        }
    }
}
```

Output:

```
[[3, 4, 5], [5, 12, 13], [6, 8, 10], [8, 15, 17], [9, 12, 15], [12, 16, 20]]
```

List rooted trees

```
func bagchain(x, n, bb, start=0) {
    n || return [x]

    var out = []
    for i in (start .. bb.end) {
        var (c, s) = bb[i]...
        if (c <= n) {
            out += bagchain([x[0] + c, x[1] + s], n-c, bb, i)
        }
    }

    return out
}

func bags(n) {
    n || return [[0, ""]]
    var upto = []
    for i in (n ^.. 1) { upto += bags(i) }
    bagchain([0, ""], n-1, upto).map{|p| [p[0]+1, '('+p[1]+')'] }
}

func replace_brackets(s) {
    var (depth, out) = (0, [])
    for c in s {
        if (c == '(') {
            out.append(<( [ {>[depth%3])
            ++depth
        }
        else {
            --depth
            out.append(<) ] }>[depth%3])
        }
    }
    return out.join
}

for x in (bags(5)) {
    say replace_brackets(x[1])
}
```

Output:

```
([{([])}])
([{()()}])
([{()}{}])
([{}{}{}])
([{()}][])
([{}{}][])
([{}][{}])
([{}][][])
([][][][])
```

Literals/Floating point

```
say 1.234
say .1234
say 1234e-5
say 12.34e5
```

Output:

```
1.234
0.1234
0.01234
1234000
```

Literals/Integer

```
say 255
say 0xff
say 0377
say 0b1111_1111
```

Output:

```
255
255
255
255
```

Literals/String

Quotes that do not interpolate:

```
'single quotes with \'embedded quote\' and \\backslash';
,unicode single quoted’;
%q(not interpolating with (nested) parentheses
and newline);
```

Quotes that interpolate:

```
var a = 42;
"double \Uquotes\E with \"embedded quote\"\nnewline and variable interpolation: #{a} % 10 = #{a %
10}";
„same as above”;
%Q(same as above);
```

Heredocs:

```
print <<EOT
Implicit double-quoted (interpolates):
a = #{a}
EOT

print <<"EOD"
Explicit double-quoted with interpolation:
a = #{a}
EOD

print <<'NON_INTERPOLATING'
This will not interpolate: #{a}
NON_INTERPOLATING
```

Logical operations

```
func logic(a, b) {
    say ("a and b: ", a && b);
    say ("a  or b: ", a || b);
    say ("a xor b: ", a ^ b);
    say ("  not a: ", !a);
}

logic(false, true);
```

Output:

```
a and b: false
a  or b: true
a xor b: true
  not a: true
```

Long multiplication

(Note that arbitrary precision arithmetic is native in Sidef).

```
say (2**64 * 2**64);
```

```
func add_with_carry(result, addend, addendpos) {
    loop {
        while (result.len < addendpos+1) {
            result.append(0)
        }
        var addend_digits = (addend.to_i + result[addendpos] -> to_s.chars)
        result[addendpos] = addend_digits.pop
        addend_digits.len > 0 || break
        addend = addend_digits.pop
        addendpos++
    }
}

func longhand_multiplication(multiplicand, multiplier) {

    var result = []
    var multiplicand_offset = 0

    multiplicand.reverse.each { |multiplicand_digit|
        var multiplier_offset = multiplicand_offset
        multiplier.reverse.each { |multiplier_digit|
            var multiplication_result = (multiplicand_digit.to_i * multiplier_digit.to_i -> to_s)

            var addend_offset = multiplier_offset
            multiplication_result.reverse.each { |result_digit_addend|
                add_with_carry(result, result_digit_addend, addend_offset)
                addend_offset++
            }
            multiplier_offset++
        }
        multiplicand_offset++
    }

    return result.join.reverse
}

say longhand_multiplication('18446744073709551616', '18446744073709551616')
```

A faster approach:

```
func long_multiplication(String a, String b) -> String {

    if (a.len < b.len) {
        (a, b) = (b, a)
    }

    '0' ~~ [a, b] && return '0'

    var x = a.reverse.chars.map{.to_n}
    var y = b.reverse.chars.map{.to_n}

    var xlen = x.end
    var ylen = y.end

    var mem = 0
    var map = y.len.of { [] }

    for j in ^y {
        for i in ^x {
            var n = (x[i]*y[j] + mem)
            var(d, m) = n.divmod(10)
            if (i == xlen) {
                map[j] << (m, d)
                mem = 0;
            }
            else {
                map[j] << m
                mem = d
            }
        }

        var n = (ylen - j)
        n > 0 && map[j].append(n.of(0)...)
        var m = (ylen - n)
        m > 0 && map[j].prepend(m.of(0)...)
    }

    var result = []
    var mrange = ^map
    var end    = (xlen + ylen + 2)

    for i in ^end {
        var n = (mrange.map {|j| map[j][i] }.sum + mem)
        (mem, result[result.end+1]) = n.divmod(10)
    }

    result.join.reverse -= /^0+/
}

say long_multiplication('18446744073709551616', '18446744073709551616')
```

Output:

```
340282366920938463463374607431768211456
```

Long primes

The smallest divisor d of p-1 such that 10^d = 1 (mod p), is the length of the period of the decimal expansion of 1/p.

```
func is_long_prime(p) {

    for d in (divisors(p-1)) {
        if (powmod(10, d, p) == 1) {
            return (d+1 == p)
        }
    }

    return false
}

say "Long primes ≤ 500:"
say primes(500).grep(is_long_prime).join(' ')

for n in ([500, 1000, 2000, 4000, 8000, 16000, 32000, 64000]) {
    say ("Number of long primes ≤ #{n}: ", primes(n).count_by(is_long_prime))
}
```

Output:

```
Long primes ≤ 500:
7 17 19 23 29 47 59 61 97 109 113 131 149 167 179 181 193 223 229 233 257 263 269 313 337 367 379 383 3
Number of long primes ≤ 500: 35
Number of long primes ≤ 1000: 60
Number of long primes ≤ 2000: 116
Number of long primes ≤ 4000: 218
Number of long primes ≤ 8000: 390
Number of long primes ≤ 16000: 716
Number of long primes ≤ 32000: 1300
Number of long primes ≤ 64000: 2430
```

Alternatively, we can implement the *is_long_prime(p)* function using the `znorder(a, p)` built-in method, which is considerably faster:

```
func is_long_prime(p) {
    znorder(10, p) == p-1
}
```

Long year?

```
func is_long_year(year) {
    Date.parse("#{year}-12-28", "%Y-%m-%d").week == 53
}

say (  "Long years in the 20th century:\n", (1900..^2000).grep(is_long_year))
say ("\nLong years in the 21st century:\n", (2000..^2100).grep(is_long_year))
say ("\nLong years in the 22nd century:\n", (2100..^2200).grep(is_long_year))
```

Output:

```
Long years in the 20th century:
[1903, 1908, 1914, 1920, 1925, 1931, 1936, 1942, 1948, 1953, 1959, 1964, 1970, 1976, 1981, 1987, 1992,

Long years in the 21st century:
[2004, 2009, 2015, 2020, 2026, 2032, 2037, 2043, 2048, 2054, 2060, 2065, 2071, 2076, 2082, 2088, 2093,

Long years in the 22nd century:
[2105, 2111, 2116, 2122, 2128, 2133, 2139, 2144, 2150, 2156, 2161, 2167, 2172, 2178, 2184, 2189, 2195]
```

Longest common prefix

```
# Finds the first point where the tree bifurcates
func find_common_prefix(hash, acc) {
    if (hash.len == 1) {
        var pair = hash.to_a[0]
        return __FUNC__(pair.value, acc+pair.key)
    }
    return acc
}

# Creates a tree like: {a => {b => {c => {}}}}
func lcp(*strings) {
    var hash = Hash()

    for str (strings.sort_by{.len}) {
        var ref = hash
        str.is_empty && return ''
        for char (str) {
            if (ref.contains(char)) {
                ref = ref{char}
                ref.len == 0 && break
            }
            else {
                ref = (ref{char} = Hash())
            }
        }
    }

    return find_common_prefix(hash, '')
}
```

Demonstration:

```
var data = [
  ["interspecies","interstellar","interstate"],
  ["throne","throne"],
  ["throne","dungeon"],
  ["throne","","throne"],
  ["cheese"],
  [""],
  [],
  ["prefix","suffix"],
  ["foo","foobar"]
]

for set in data {
    say "lcp(#{set.dump.substr(1,-1)}) = #{lcp(set...).dump}"
}
```

Output:

```
lcp("interspecies", "interstellar", "interstate") = "inters"
lcp("throne", "throne") = "throne"
lcp("throne", "dungeon") = ""
lcp("throne", "", "throne") = ""
lcp("cheese") = "cheese"
lcp("") = ""
lcp() = ""
lcp("prefix", "suffix") = ""
lcp("foo", "foobar") = "foo"
```

Longest common subsequence

```
func lcs(xstr, ystr) is cached {

    xstr.is_empty && return xstr
    ystr.is_empty && return ystr

    var(x, xs, y, ys) = (xstr.ft(0,0), xstr.ft(1),
                         ystr.ft(0,0), ystr.ft(1))

    if (x == y) {
        x + lcs(xs, ys)
    } else {
        [lcs(xstr, ys), lcs(xs, ystr)].max_by { .len }
    }
}

say lcs("thisisatest", "testing123testing")
```

Output:

```
tsitest
```

Longest common substring

```
func createSubstrings(String word) -> Array {
  gather {
    combinations(word.len+1, 2, {|i,j|
        take(word.substr(i, j-i))
    })
  }
}

func findLongestCommon(String first, String second) -> String {
    createSubstrings(first) & createSubstrings(second) -> max_by { .len }
}

say findLongestCommon("thisisatest", "testing123testing")
```

Output:

```
test
```

Longest increasing subsequence

Dynamic programming:

```
func lis(a) {
    var l = a.len.of { [] }
    l[0] << a[0]
    for i in (1..a.end) {
        for j in ^i {
            if ((a[j] < a[i]) && (l[i].len < l[j].len+1)) {
                l[i] = [l[j]...]
            }
        }
        l[i] << a[i]
    }
    l.max_by { .len }
}

say lis(%i<3 2 6 4 5 1>)
say lis(%i<0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15>)
```

Patience sorting:

```
func lis(deck) {
    var pileTops = []
    deck.each { |x|
        var low = 0;
        var high = pileTops.end
        while (low <= high) {
            var mid = ((low + high) // 2)
            if (pileTops[mid]{:val} >= x) {
                high = mid-1
            } else {
                low = mid+1
            }
        }
        var i = low
        var node = Hash(val => x)
        node{:back} = pileTops[i-1] if (i != 0)
        pileTops[i] = node
    }
    var result = []
    for (var node = pileTops[-1]; node; node = node{:back}) {
        result << node{:val}
    }
    result.reverse
}

say lis(%i<3 2 6 4 5 1>)
say lis(%i<0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15>)
```

Output:

```
[2, 4, 5]
[0, 2, 6, 9, 11, 15]
```

Longest string challenge

```
var l = ''  # Sample longest string seen.
var a = ''  # Accumulator to save longest strings.

STDIN.each { |n|
    n.substr(l.len) ? (a = n; l = n)
                    : (!l.substr(n.len) && a.concat!(n))
}

print a
```

Look-and-say sequence

```
func lookandsay(str) {
    str.gsub(/((.)\2*)/, {|a,b| a.len.to_s + b })
}

var num = "1"
{
  say num;
  num = lookandsay(num)
} * 10
```

Output:

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
```

Loop over multiple arrays simultaneously

The simplest way is by using the `Array.zip()` method:

```
[%w(a b c),%w(A B C),%w(1 2 3)].zip { |i,j,k|
    say (i, j, k)
}
```

Output:

```
aA1
bB2
cC3
```

Loops/Break

```
var lim = 20
loop {
    say (var n = lim.irand)
    n == 10 && break
    say lim.irand
}
```

Loops/Continue

```
for i in (1..10) {
    print i
    if (i % 5 == 0) {
        print "\n"
        next
    }
    print ', '
}
```

Loops/Do-while

```
var value = 0
do {
    say ++value
} while (value % 6)
```

Loops/Downward for

for(;;) loop:

```
for (var i = 10; i >= 0; i--) {
    say i
}
```

for-in loop:

```
for i in (11 ^.. 0) {
    say i
}
```

.each method:

```
10.downto(0).each { |i|
    say i
}
```

Loops/For

for(;;) loop:

```
for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= i; j++) {
        print '*'
    }
    print "\n"
}
```

for([]) loop:

```
for (1..5) { |i|
    for (1..i) { print '*' }
    print "\n"
}
```

for-in loop:

```
for i in (1..5) {
    for j in (1..i) { print '*' }
    print "\n"
}
```

Loops/For with a specified step

for(;;) loop:

```
for (var i = 2; i <= 8; i += 2) {
    say i
}
```

for-in loop:

```
for i in (2 .. (8, 2)) {
    say i
}
```

.each method:

```
2.to(8).by(2).each { |i|
    say i
}
```

Loops/Foreach

foreach loop:

```
foreach [1,2,3] { |i|
    say i
}
```

for-in loop:

```
for i in [1,2,3] {
    say i
}
```

.each method:

```
[1,2,3].each { |i|
    say i
}
```

Loops/Infinite

```
loop { say "SPAM!" }
```

Loops/N plus one half

```
for i in (1..10) {
    print i
    i == 10 && break
    print ', '
}

print "\n"
```

Loops/Nested

```
var arr = 10.of{ 10.of{ pick(1..20) } }

for row in arr {
    for num in row {
        "%3d".printf(num)
        num == 20 && goto :OUT
    }
    print "\n"
} @:OUT

print "\n"
```

Output:

```
  9 17 14 17 17  7  1  3  9 18
  1 12  1 19  9  5  1 17 19  3
 17  2 18 12 15 10  8 13 13 14
 12 16 13 13  2 11  3 15  2  4
 15 15  8 11  5  2  1 16  8 13
 17  3  1  1  8 12  4 20
```

Loops/While

```
var i = 1024
while (i > 0) {
    say i
    i //= 2
}
```

LU decomposition

```
func is_square(m) { m.all { .len == m.len } }
func matrix_zero(n, m=n) { m.of { n.of(0) } }
func matrix_ident(n) { n.of {|i| [i.of(0)..., 1, (n - i - 1).of(0)...] } }

func pivotize(m) {
    var size = m.len
    var id = matrix_ident(size)
    for i in (^size) {
        var max = m[i][i]
        var row = i
        for j in (i ..^ size) {
            if (m[j][i] > max) {
                max = m[j][i]
                row = j
            }
        }
        if (row != i) {
            id.swap(row, i)
        }
    }
    return id
}

func mmult(a, b) {
    var p = []
    for r in ^a, c in ^b[0], i in ^b {
        p[r][c] := 0 += (a[r][i] * b[i][c])
    }
    return p
}

func lu(a) {
    is_square(a) || die "Defined only for square matrices!";
    var n = a.len
    var P = pivotize(a)
    var A' = mmult(P, a)
    var L = matrix_ident(n)
    var U = matrix_zero(n)
    for i in ^n, j in ^n {
        if (j >= i) {
            U[i][j] = (A'[i][j] - sum(^i, { U[_][j] * L[i][_] }))
        } else {
            L[i][j] = ((A'[i][j] - sum(^j, { U[_][j] * L[i][_] })) / U[j][j])
        }
    }
    return [P, A', L, U]
}

func say_it(message, array) {
    say "\n#{message}"
    array.each { |row|
        say row.map{"%7s" % .as_rat}.join(' ')
    }
```

```
}

var t = [[
   %n(1 3 5),
   %n(2 4 7),
   %n(1 1 0),
],[
   %n(11  9 24  2),
   %n( 1  5  2  6),
   %n( 3 17 18  1),
   %n( 2  5  7  1),
]]

t.each { |test|
    say_it('A Matrix', test)
    for a,b in (['P Matrix', 'A' Matrix', 'L Matrix', 'U Matrix'] ~Z lu(test)) {
        say_it(a, b)
    }
}
```

Lucas-Lehmer test

```
func is_mersenne_prime(p) {
    return true if (p == 2)
    var s = 4
    var M = (2**p - 1)
    { s = powmod(s, 2, M)-2 } * (p-2)
    s == 0
}

Inf.times {|n|
    if (n.is_prime && is_mersenne_prime(n)) {
        say "M#{n}"
    }
}
```

Output:

```
M2
M3
M5
M7
M13
M17
M19
M31
M61
M89
M107
M127
M521
M607
M1279
M2203
M2281
^C
```

Ludic numbers

```
func ludics_upto(nmax=100000) {
  Enumerator({ |collect|
    collect(1)
    var arr = @(2..nmax)
    while (arr) {
      collect(var n = arr[0])
      {|i| arr[i] = nil} << (^arr `by` n)
      arr.compact!
    }
  })
}

func ludics_first(n) {
    ludics_upto(n * n.log2).first(n)
}

say("First 25 Ludic numbers: ",     ludics_first(25).join(' '))
say("Ludics below 1000: ",          ludics_upto(1000).len)
say("Ludic numbers 2000 to 2005: ", ludics_first(2005).last(6).join(' '))

var a = ludics_upto(250).to_a
say("Ludic triples below 250: ", a.grep{|x| a.contains_all([x+2, x+6]) } \
                                 .map {|x| '(' + [x, x+2, x+6].join(' ') + ')' } \
                                 .join(' '))
```

Output:

```
First 25 Ludic numbers: 1 2 3 5 7 11 13 17 23 25 29 37 41 43 47 53 61 67 71 77 83 89 91 97 107
Ludics below 1000: 142
Ludic numbers 2000 to 2005: 21475 21481 21487 21493 21503 21511
Ludic triples below 250: (1 3 7) (5 7 11) (11 13 17) (23 25 29) (41 43 47) (173 175 179) (221 223 227)
```

Luhn test of credit card numbers

```
func luhn (n) {
    static a = {|j| (2*j // 10) + (2*j % 10) }.map(^10)

    var checksum = n.digits.map_kv {|i,j|
        i.is_odd ? a[j] : j
    }.sum

    checksum % 10 == 0
}

for n in [49927398716, 49927398717, 1234567812345678, 1234567812345670] {
    say [n, luhn(n)]
}
```

Output:

```
[49927398716, true]
[49927398717, false]
[1234567812345678, false]
[1234567812345670, true]
```

Lychrel numbers

```
var (
    lychrels = [],
    palindromes = [],
    seeds = [],
    max = 500,
)

for i in (1 .. 10_000) {
    var (test = [], count = 0)

    func lychrel(l) {
        count++ > max && return true
        test << (var m = (l + l.flip))
        m.is_palindrome && return false
        lychrel(m)
    }

    if (lychrel(i)) {
        lychrels << Pair(Str(i), test)
    }
}

seeds << lychrels[0]

for l in lychrels {
    if (l.key.is_palindrome) {
        palindromes << l.key
    }

    var h = Hash()
    h.set_keys(l.value...)

    var trial = seeds.count_by { |s|
        s.value.any { |k| h.contains(k) } ? break : true
    }

    if (trial == seeds.len) {
        seeds << l
    }
}

say ("   Number of Lychrel seed numbers < 10_000: ", seeds.len)
say ("             Lychrel seed numbers < 10_000: ", seeds.map{.key}.join(', '))
say ("Number of Lychrel related numbers < 10_000: ", lychrels.len - seeds.len)
say ("    Number of Lychrel palindromes < 10_000: ", palindromes.len)
say ("              Lychrel palindromes < 10_000: ", palindromes.join(', '))
```

Output:

```
   Number of Lychrel seed numbers < 10_000: 5
             Lychrel seed numbers < 10_000: 196, 879, 1997, 7059, 9999
Number of Lychrel related numbers < 10_000: 244
    Number of Lychrel palindromes < 10_000: 3
              Lychrel palindromes < 10_000: 4994, 8778, 9999
```

LZW compression

```
# Compress a string to a list of output symbols.
func compress(String uncompressed) -> Array {

    # Build the dictionary.
    var dict_size = 256
    var dictionary = Hash()

    for i in range(dict_size) {
        dictionary{i.chr} = i.chr
    }

    var w = ''
    var result = []
    uncompressed.each { |c|
        var wc = w+c
        if (dictionary.exists(wc)) {
            w = wc
        } else {
            result << dictionary{w}
            # Add wc to the dictionary.
            dictionary{wc} = dict_size
            dict_size++
            w = c
        }
    }

    # Output the code for w.
    if (w != '') {
        result << dictionary{w}
    }

    return result
}

# Decompress a list of output ks to a string.
func decompress(Array compressed) -> String {

    # Build the dictionary.
    var dict_size = 256
    var dictionary = Hash()

    for i in range(dict_size) {
        dictionary{i.chr} = i.chr
    }

    var w = compressed.shift
    var result = w
    compressed.each { |k|
        var entry = nil
        if (dictionary.exists(k)) {
            entry = dictionary{k}
        } elsif (k == dict_size) {
            entry = w+w.first
```

```
        } else {
            die "Bad compressed k: #{k}"
        }
        result += entry

        # Add w+entry[0] to the dictionary.
        dictionary{dict_size} = w+entry.first
        dict_size++

        w = entry
    }
    return result
}

# How to use:
var compressed = compress('TOBEORNOTTOBEORTOBEORNOT')
say compressed.join(' ')
var decompressed = decompress(compressed)
say decompressed
```

Output:

```
T O B E O R N O T 256 258 260 265 259 261 263
TOBEORNOTTOBEORTOBEORNOT
```

Möbius function

Built-in:

```
say moebius(53)    #=> -1
say moebius(54)    #=> 0
say moebius(55)    #=> 1
```

Simple implementation:

```
func μ(n) {
    var f = n.factor_exp.map { .tail }
    f.any { _ > 1 } ? 0 : ((-1)**f.sum)
}

with (199) { |n|
    say "Values of the Möbius function for numbers in the range 1..#{n}:"
    [' '] + (1..n->map(μ)) -> each_slice(20, {|*line|
        say line.map { '%2s' % _ }.join(' ')
    })
}
```

Output:

```
Values of the Möbius function for numbers in the range 1..199:
    1 -1 -1  0 -1  1 -1  0  0  1 -1  0 -1  1  1  0 -1  0 -1
 0  1  1 -1  0  0  1  0  0 -1 -1 -1  0  1  1  1  0 -1  1  1
 0 -1 -1 -1  0  0  1 -1  0  0  0  1  0 -1  0  1  0  1  1 -1
 0 -1  1  0  0  1 -1 -1  0  1 -1 -1  0 -1  1  0  0  1 -1 -1
 0  0  1 -1  0  1  1  1  0 -1  0  1  0  1  1  1  0 -1  0  0
 0 -1 -1 -1  0 -1  1 -1  0 -1 -1  1  0 -1 -1  1  0  0  1  1
 0  0  1  1  0  0  0 -1  0  1 -1 -1  0  1  1  0  0 -1 -1 -1
 0  1  1  1  0  1  1  0  0 -1  0 -1  0  0 -1  1  0 -1  1  1
 0  1  0 -1  0 -1  1 -1  0  0 -1  0  0 -1 -1  0  0  1  1 -1
 0 -1 -1  1  0  1 -1  1  0  0 -1 -1  0 -1  1 -1  0 -1  0 -1
```

Mad Libs

```
var story = ARGF.slurp;

var blanks = Hash.new;
while (var m = /<(.*?)>/.gmatch(story)) {
    blanks.append(m[0]);
}

blanks.keys.sort.each { |blank|
    var replacement = Sys.scanln("#{blank}: ");
    blanks{blank} = replacement;
}

print story.gsub(/<(.*?)>/, {|s1| blanks{s1} });
```

Magic constant

```
func f(n) {
    (n+2) * ((n+2)**2 + 1) / 2
}

func order(n) {
    iroot(2*n, 3) + 1
}

say ("First 20 terms: ", f.map(1..20).join(' '))
say ("1000th term: ", f(1000), " with order ", order(f(1000)))

for n in (1 .. 20) {
    printf("order(10^%-2s) = %s\n", n, order(10**n))
}
```

Output:

```
First 20 terms: 15 34 65 111 175 260 369 505 671 870 1105 1379 1695 2056 2465 2925 3439 4010 4641 5335
1000th term: 503006505 with order 1003
order(10^1 ) = 3
order(10^2 ) = 6
order(10^3 ) = 13
order(10^4 ) = 28
order(10^5 ) = 59
order(10^6 ) = 126
order(10^7 ) = 272
order(10^8 ) = 585
order(10^9 ) = 1260
order(10^10) = 2715
order(10^11) = 5849
order(10^12) = 12600
order(10^13) = 27145
order(10^14) = 58481
order(10^15) = 125993
order(10^16) = 271442
order(10^17) = 584804
order(10^18) = 1259922
order(10^19) = 2714418
order(10^20) = 5848036
```

Magic squares of doubly even order

```
func double_even_magic_square(n) {
    assert(n%4 == 0, "Need multiple of four")
    var (bsize, max) = (n/4, n*n)
    var pre_pat = [true, false, false, true,
                   false, true, true, false]
    pre_pat += pre_pat.flip
    var pattern = (pre_pat.map{|b| bsize.of(b)... } * bsize)
    pattern.map_kv{|k,v| v ? k+1 : max-k }.slices(n)
}

func format_matrix(a) {
    var fmt = "%#{a.len**2 -> len}s"
    a.map { .map { fmt % _ }.join(' ') }.join("\n")
}

say format_matrix(double_even_magic_square(8))
```

Output:

```
 1  2 62 61 60 59  7  8
56 55 11 12 13 14 50 49
48 47 19 20 21 22 42 41
25 26 38 37 36 35 31 32
33 34 30 29 28 27 39 40
24 23 43 44 45 46 18 17
16 15 51 52 53 54 10  9
57 58  6  5  4  3 63 64
```

Magic squares of odd order

```
func magic_square(n {.is_pos && .is_odd}) {
    var i = 0
    var j = int(n/2)

    var magic_square = []
    for l in (1 .. n**2) {
        magic_square[i][j] = l

        if (magic_square[i.dec % n][j.inc % n]) {
            i = (i.inc % n)
        }
        else {
            i = (i.dec % n)
            j = (j.inc % n)
        }
    }

    return magic_square
}

func print_square(sq) {
    var f = "%#{(sq.len**2).len}d";
    for row in sq {
        say row.map{ f % _ }.join(' ')
    }
}

var(n=5) = ARGV»to_i()»...
var sq = magic_square(n)
print_square(sq)

say "\nThe magic number is: #{sq[0].sum}"
```

Output:

```
17 24  1  8 15
23  5  7 14 16
 4  6 13 20 22
10 12 19 21  3
11 18 25  2  9

The magic number is: 65
```

Magnanimous numbers

```
func is_magnanimous(n) {
    1..n.ilog10 -> all {|k|
        sum(divmod(n, k.ipow10)).is_prime
    }
}

say "First 45 magnanimous numbers:"
say is_magnanimous.first(45).join(' ')

say "\n241st through 250th magnanimous numbers:"
say is_magnanimous.first(250).last(10).join(' ')

say "\n391st through 400th magnanimous numbers:"
say is_magnanimous.first(400).last(10).join(' ')
```

Output:

```
First 45 magnanimous numbers:
0 1 2 3 4 5 6 7 8 9 11 12 14 16 20 21 23 25 29 30 32 34 38 41 43 47 49 50 52 56 58 61 65 67 70 74 76 83

241st through 250th magnanimous numbers:
17992 19972 20209 20261 20861 22061 22201 22801 22885 24407

391st through 400th magnanimous numbers:
486685 488489 515116 533176 551558 559952 595592 595598 600881 602081
```

Make directory path

```
Dir(Dir.cwd, "path", "to", "dir").make_path   # works cross-platform
```

Man or boy test

```
func a(k, x1, x2, x3, x4, x5) {
    func b { a(--k, b, x1, x2, x3, x4) }
    k <= 0 ? (x4() + x5()) : b()
}
say a(10, ->{1}, ->{-1}, ->{-1}, ->{1}, ->{0})      #=> -67
```

This solution avoids creating the closure b if k <= 0 (that is, nearly every time).

```
func a(k, x1, x2, x3, x4, x5) {
    k <= 0 ? (x4() + x5())
           : func b { a(--k, b, x1, x2, x3, x4) }()
}
say a(10, ->{1}, ->{-1}, ->{-1}, ->{1}, ->{0})      #=> -67
```

Alternatively, we can implement it as a method too:

```
class MOB {
    method a(k, x1, x2, x3, x4, x5) {
        func b { self.a(--k, b, x1, x2, x3, x4) }
        k <= 0 ? (x4() + x5()) : b()
    }
}

var obj = MOB()
say obj.a(10, ->{1}, ->{-1}, ->{-1}, ->{1}, ->{0})
```

Mandelbrot set

```
func mandelbrot(z) {
    var c = z
    {   z = (z*z + c)
        z.abs > 2 && return true
    } * 20
    return false
}

for y range(1, -1, -0.05) {
    for x in range(-2, 0.5, 0.0315) {
        print(mandelbrot(x + y.i) ? ' ' : '#')
    }
    print "\n"
}
```

Map range

```
func map_range(a, b, x) {
    var (a1, a2, b1, b2) = (a.bounds, b.bounds)
    x-a1 * b2-b1 / a2-a1 + b1
}

var a = 0..10
var b = -1..0

a.each { |x|
    say "#{x} maps to #{map_range(a, b, x)}"
}
```

Output:

```
0 maps to -1
1 maps to -0.9
2 maps to -0.8
3 maps to -0.7
4 maps to -0.6
5 maps to -0.5
6 maps to -0.4
7 maps to -0.3
8 maps to -0.2
9 maps to -0.1
10 maps to 0
```

Markov chain text generator

```
func build_dict (n, words) {
    var dict = Hash()
    words.each_cons(n+1, {|*prefix|
        var suffix = prefix.pop
        dict{prefix.join(' ')} := [] << suffix
    })
    return dict
}

var file = File(ARGV[0] || "alice_oz.txt")
var n    =  Num(ARGV[1] || 2)
var max  =  Num(ARGV[2] || 100)

var words = file.open_r.words
words << words.first(n)...

var dict = build_dict(n, words)
var rotor = words.first(n)
var chain = [rotor...]

max.times {
    var new = dict{rotor.join(' ')}.rand
    chain.push(new)
    rotor.shift
    rotor.push(new)
}

say chain.join(' ')
```

Output:

```
Alice was a large caterpillar, that was linked into hers began to cry a little startled when she knows
```

Matrix-exponentiation operator

```
class Array {
    method ** (Number n { .>= 0 }) {
        var tmp = self
        var out = self.len.of {|i| self.len.of {|j| i == j ? 1 : 0 }}
        loop {
            out = (out `mmul` tmp) if n.is_odd
            n >>= 1 || break
            tmp = (tmp `mmul` tmp)
        }
        return out
    }
}

var m = [[1, 2, 0],
         [0, 3, 1],
         [1, 0, 0]]

for order in (0..5) {
    say "### Order #{order}"
    var t = (m ** order)
    say ('  ', t.join("\n  "))
}
```

Output:

```
### Order 0
  [1, 0, 0]
  [0, 1, 0]
  [0, 0, 1]
### Order 1
  [1, 2, 0]
  [0, 3, 1]
  [1, 0, 0]
### Order 2
  [1, 8, 2]
  [1, 9, 3]
  [1, 2, 0]
### Order 3
  [3, 26, 8]
  [4, 29, 9]
  [1, 8, 2]
### Order 4
  [11, 84, 26]
  [13, 95, 29]
  [3, 26, 8]
### Order 5
  [37, 274, 84]
  [42, 311, 95]
  [11, 84, 26]
```

Matrix multiplication

```
func matrix_multi(a, b) {
    var m = [[]]
    for r in ^a {
        for c in ^b[0] {
            for i in ^b {
                m[r][c] := 0 += (a[r][i] * b[i][c])
            }
        }
    }
    return m
}

var a = [
          [1, 2],
          [3, 4],
          [5, 6],
          [7, 8]
        ]

var b = [
          [1, 2, 3],
          [4, 5, 6]
        ]

for line in matrix_multi(a, b) {
    say line.map{|i|'%3d' % i }.join(', ')
}
```

Output:

```
  9,  12,  15
 19,  26,  33
 29,  40,  51
 39,  54,  69
```

Matrix transposition

```
func transpose(matrix) {
    matrix[0].range.map {|i| matrix.map { _[i] } }
}

var m = [
  [1,  1,   1,   1],
  [2,  4,   8,  16],
  [3,  9,  27,  81],
  [4, 16,  64, 256],
  [5, 25, 125, 625],
]

transpose(m).each { |row|
    "%5d" * row.len -> printlnf(row...)
}
```

Output:

```
    1    2    3    4    5
    1    4    9   16   25
    1    8   27   64  125
    1   16   81  256  625
```

Matrix with two diagonals

```
func dual_diagonal(n) {
    n.of {|k|
        var r = (k.of(0) + [1] + (n - k - 1).of(0))
        r ~Z| r.reverse
    }
}

dual_diagonal(5).each{.join(' ').say}; say ''
dual_diagonal(6).each{.join(' ').say}
```

Output:

```
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1

1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
0 0 1 1 0 0
0 1 0 0 1 0
1 0 0 0 0 1
```

Maximum triangle path sum

Iterative solution:

```
var sum = [0]

ARGF.each {  |line|
    var x = line.words.map{.to_n}
    sum = [
            x.first + sum.first,
            {|i| x[i] + [sum[i-1, i]].max }.map(1 ..^ x.end)...
            x.last + sum.last,
          ]
}

say sum.max
```

Recursive solution:

```
var triangle = ARGF.slurp.lines.map{.words.map{.to_n}}

func max_value(i=0, j=0) is cached {
    i == triangle.len && return 0
    triangle[i][j] + [max_value(i+1, j), max_value(i+1, j+1)].max
}

say max_value()
```

Output:

```
% sidef maxpath.sf triangle.txt
1320
```

Maze generation

```
var(w:5, h:5) = ARGV.map{.to_i}...
var avail = (w * h)

# cell is padded by sentinel col and row, so I don't check array bounds
var cell = (1..h -> map {([true] * w) + [false]} + [[false] * w+1])
var ver  = (1..h -> map {["|  "] * w })
var hor  = (0..h -> map {["+--"] * w })

func walk(x, y) {
    cell[y][x] = false;
    avail-- > 0 || return nil;  # no more bottles, er, cells

    var d = [[-1, 0], [0, 1], [1, 0], [0, -1]]
    while (!d.is_empty) {
        var i = d.pop_rand
        var (x1, y1) = (x + i[0], y + i[1])

        cell[y1][x1] || next

        if (x == x1) { hor[[y1, y].max][x] = '+  ' }
        if (y == y1) { ver[y][[x1, x].max] = '   ' }
        walk(x1, y1)
    }
}

walk(w.rand.int, h.rand.int)   # generate

for i in (0 .. h) {
    say (hor[i].join('') + '+')
    if (i < h) {
        say (ver[i].join('') + '|')
    }
}
```

Output:

```
+--+--+--+--+--+
|           |  |
+--+  +--+  +  +
|     |        |
+  +--+--+--+--+
|     |        |
+--+  +  +--+  +
|     |  |     |
+  +--+--+  +--+
|              |
+--+--+--+--+--+
```

MD4

```
var digest = frequire('Digest::MD4')
say digest.md4_hex('Rosetta Code')
```

Output:

```
a52bcfc6a0d0d300cdc5ddbfbefe478b
```

MD5

```
var digest = frequire('Digest::MD5')
say digest.md5_hex("The quick brown fox jumped over the lazy dog's back")
```

The same in OO manner

```
var md5 = require('Digest::MD5').new
md5.add("The quick brown fox jumped over the lazy dog's back")
say md5.hexdigest
```

MD5/Implementation

```
class MD5(String msg) {

    method init {
        msg = msg.bytes
    }

    const FGHI = [
        {|a,b,c| (a & b) | (~a & c) },
        {|a,b,c| (a & c) | (b & ~c) },
        {|a,b,c| (a ^ b ^ c)        },
        {|a,b,c| (b ^ (a | ~c))     },
    ]

    const S = [
        [7, 12, 17, 22] * 4,
        [5,  9, 14, 20] * 4,
```

```
        [4, 11, 16, 23] * 4,
        [6, 10, 15, 21] * 4,
    ].flat

    const T = 64.of {|i| floor(abs(sin(i+1)) * 1<<32) }

    const K = [
        ^16 -> map {|n|    n           },
        ^16 -> map {|n| (5*n + 1) % 16 },
        ^16 -> map {|n| (3*n + 5) % 16 },
        ^16 -> map {|n| (7*n    ) % 16 },
    ].flat

    func radix(Number b, Array a) {
        ^a -> sum {|i| b**i * a[i] }
    }

    func little_endian(Number w, Number n, Array v) {
        var step1 = (^n »*» w)
        var step2 = (v ~X>> step1)
        step2 »%» (1 << w)
    }

    func block(Number a, Number b) { (a  + b) & 0xffffffff }
    func srble(Number a, Number n) { (a << n) & 0xffffffff | (a >> (32-n)) }

    func md5_pad(msg) {
        var bits = 8*msg.len
        var padded = [msg..., 128, [0] * (-(floor(bits / 8) + 1 + 8) % 64)].flat

        gather {
            padded.each_slice(4, {|*a|
                take(radix(256, a))
            })
            take(little_endian(32, 2, [bits]))
        }.flat
    }

    func md5_block(Array H, Array X) {
        var (A, B, C, D) = H...

        for i in ^64 {
            (A, B, C, D) = (D,
                block(B, srble(
                    block(
                        block(
                            block(A, FGHI[floor(i / 16)](B, C, D)), T[i]
                        ), X[K[i]]
                    ), S[i])
                ), B, C)
        }

        for k,v in ([A, B, C, D].kv) {
            H[k] = block(H[k], v)
        }

        return H
    }

    method md5_hex {
        self.md5.map {|n| '%02x' % n }.join
    }

    method md5 {
        var M = md5_pad(msg)
        var H = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]
```

```
        var H = [0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476]

        for i in (range(0, M.end, 16)) {
            md5_block(H, M.ft(i, i+15))
        }

        little_endian(8, 4, H)
    }
}

var tests = [
    ['d41d8cd98f00b204e9800998ecf8427e', ''],
    ['0cc175b9c0f1b6a831c399e269772661', 'a'],
    ['900150983cd24fb0d6963f7d28e17f72', 'abc'],
    ['f96b697d7cb7938d525a2f31aaf161d0', 'message digest'],
    ['c3fcd3d76192e4007dfb496cca67e13b', 'abcdefghijklmnopqrstuvwxyz'],
    ['d174ab98d277d9f5a5611c2c9f419d9f',
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'],
    ['57edf4a22be3c955ac49da2e2107b67a',
'12345678901234567890123456789012345678901234567890123456789012345678901234567890'],
]

for md5,msg in tests {
    var hash = MD5(msg).md5_hex
    say "#{hash} : #{msg}"

    if (hash != md5) {
        say "\tHowever, that is incorrect (expected: #{md5})"
    }
}
```

Output:

```
d41d8cd98f00b204e9800998ecf8427e :
0cc175b9c0f1b6a831c399e269772661 : a
900150983cd24fb0d6963f7d28e17f72 : abc
f96b697d7cb7938d525a2f31aaf161d0 : message digest
c3fcd3d76192e4007dfb496cca67e13b : abcdefghijklmnopqrstuvwxyz
d174ab98d277d9f5a5611c2c9f419d9f : ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
57edf4a22be3c955ac49da2e2107b67a : 12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Menu

```
func menu (prompt, arr) {
    arr.len > 0 || return ''
    loop {
        for i in ^arr {
            say "  #{i}: #{arr[i]}"
        }
        var n = Sys.scanln(prompt) \\ return()
        n ~~ /^[0-9]+\z/ ? Num(n) : next
        arr.exists(n) && return arr[n]
    }
}

var list = ['fee fie', 'huff and puff', 'mirror mirror', 'tick tock']
var prompt = 'Please choose an item number: '

var answer = menu(prompt, list)
say "You choose: #{answer}"
```

Mersenne primes

Uses the *is_mersenne_prime()* function from Math::Prime::Util::GMP.

```
for p in (^Inf -> lazy.grep { .is_mersenne_prime }) {
    say "2^#{p} - 1"
}
```

Output:

```
2^2 - 1
2^3 - 1
2^5 - 1
2^7 - 1
2^13 - 1
2^17 - 1
2^19 - 1
2^31 - 1
2^61 - 1
2^89 - 1
2^107 - 1
2^127 - 1
2^521 - 1
2^607 - 1
2^1279 - 1
2^2203 - 1
2^2281 - 1
2^3217 - 1
2^4253 - 1
2^4423 - 1
2^9689 - 1
2^9941 - 1
^C
sidef mersenne.sf  12.47s user 0.02s system 99% cpu 12.495 total
```

Mertens function

Built-in:

```
say mertens(123456789)   #=> 1170
say mertens(1234567890)  #=> 9163
```

Algorithm for computing M(n) in sublinear time:

```
func mertens(n) is cached {

    var lookup_size    = (2 * n.iroot(3)**2)
    var mertens_lookup = [0]

    for k in (1..lookup_size) {
        mertens_lookup[k] = (mertens_lookup[k-1] + k.moebius)
    }

    static cache = Hash()

    func (n) {

        if (n <= lookup_size) {
            return mertens_lookup[n]
        }

        if (cache.has(n)) {
            return cache{n}
        }

        var M = 1
        var s = n.isqrt

        for k in (2 .. floor(n/(s+1))) {
            M -= __FUNC__(floor(n/k))
        }

        for k in (1..s) {
            M -= (mertens_lookup[k] * (floor(n/k) - floor(n/(k+1))))
        }

        cache{n} = M
    }(n)
}
```

Task:

```
with (200) {|n|
    say "Mertens function in the range 1..#{n}:"
    (1..n).map { mertens(_) }.slices(20).each {|line|
        say line.map{ "%2s" % _ }.join(' ')
    }
}

with (1000) {|n|
    say "\nIn the range 1..#{n}, there are:"
    say (1..n->count_by { mertens(_)==0 }, " zeros")
    say (1..n->count_by { mertens(_)==0 && mertens(_-1)!=0 }, " zero crossings")
}
```

Output:

```
Mertens function in the range 1..200:
 1  0 -1 -1 -2 -1 -2 -2 -2 -1 -2 -2 -3 -2 -1 -1 -2 -2 -3 -3
-2 -1 -2 -2 -2 -1 -1 -1 -2 -3 -4 -4 -3 -2 -1 -1 -2 -1  0  0
-1 -2 -3 -3 -3 -2 -3 -3 -3 -3 -2 -2 -3 -3 -2 -2 -1  0 -1 -1
-2 -1 -1 -1  0 -1 -2 -2 -1 -2 -3 -3 -4 -3 -3 -3 -2 -3 -4 -4
-4 -3 -4 -4 -3 -2 -1 -1 -2 -2 -1 -1  0  1  2  2  1  1  1  1
 0 -1 -2 -2 -3 -2 -3 -3 -4 -5 -4 -4 -5 -6 -5 -5 -5 -4 -3 -3
-3 -2 -1 -1 -1 -1 -2 -2 -1 -2 -3 -3 -2 -1 -1 -1 -2 -3 -4 -4
-3 -2 -1 -1  0  1  1  1  0  0 -1 -1 -1 -2 -1 -1 -2 -1  0  0
 1  1  0  0 -1  0 -1 -1 -1 -2 -2 -2 -3 -4 -4 -4 -3 -2 -3 -3
-4 -5 -4 -4 -3 -4 -3 -3 -3 -4 -5 -5 -6 -5 -6 -6 -7 -7 -8 -8

In the range 1..1000, there are:
92 zeros
59 zero crossings
```

Metallic ratios

```
func seqRatio(f, places = 32) {
    1..Inf -> reduce {|t,n|
        var r = (f(n+1)/f(n)).round(-places)
        return(n, r.as_dec(places + r.abs.int.len)) if (r == t)
        r
    }
}

for k,v in (%w(Platinum Golden Silver Bronze Copper Nickel Aluminum Iron Tin Lead).kv) {
    next if (k == 0)   # undefined ratio
    say "Lucas sequence U_n(#{k},-1) for #{v} ratio"
    var f = {|n| lucasu(k, -1, n) }
    say ("First 15 elements: ", 15.of(f).join(', '))
    var (n, r) = seqRatio(f)
    say "Approximated value: #{r} reached after #{n} iterations"
    say ''
}

with (seqRatio({|n| fib(n) }, 256)) {|n,v|
    say "Golden ratio to 256 decimal places:"
    say "Approximated value: #{v}"
    say "Reached after #{n} iterations"
}
```

Output:

```
Lucas sequence U_n(1,-1) for Golden ratio
First 15 elements: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377
Approximated value: 1.6180339887498948482045868343656 reached after 79 iterations

Lucas sequence U_n(2,-1) for Silver ratio
First 15 elements: 0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461, 80782
Approximated value: 2.4142135623730950488016887242097 reached after 45 iterations

Lucas sequence U_n(3,-1) for Bronze ratio
First 15 elements: 0, 1, 3, 10, 33, 109, 360, 1189, 3927, 12970, 42837, 141481, 467280, 1543321, 509724
Approximated value: 3.3027756377319946465596106337352 reached after 33 iterations

Lucas sequence U_n(4,-1) for Copper ratio
First 15 elements: 0, 1, 4, 17, 72, 305, 1292, 5473, 23184, 98209, 416020, 1762289, 7465176, 31622993,
Approximated value: 4.2360679774997896964091736687312 reached after 28 iterations

Lucas sequence U_n(5,-1) for Nickel ratio
First 15 elements: 0, 1, 5, 26, 135, 701, 3640, 18901, 98145, 509626, 2646275, 13741001, 71351280, 3704
Approximated value: 5.1925824035672520156253552457701 6 reached after 26 iterations

Lucas sequence U_n(6,-1) for Aluminum ratio
First 15 elements: 0, 1, 6, 37, 228, 1405, 8658, 53353, 328776, 2026009, 12484830, 76934989, 474094764,
Approximated value: 6.1622776601683793319988935444327 2 reached after 23 iterations

Lucas sequence U_n(7,-1) for Iron ratio
First 15 elements: 0, 1, 7, 50, 357, 2549, 18200, 129949, 927843, 6624850, 47301793, 337737401, 2411463
Approximated value: 7.1400549446402591355486512457635 2 reached after 21 iterations

Lucas sequence U_n(8,-1) for Tin ratio
First 15 elements: 0, 1, 8, 65, 528, 4289, 34840, 283009, 2298912, 18674305, 151693352, 1232221121, 100
Approximated value: 8.1231056256176605498214098559740 8 reached after 20 iterations

Lucas sequence U_n(9,-1) for Lead ratio
First 15 elements: 0, 1, 9, 82, 747, 6805, 61992, 564733, 5144589, 46866034, 426938895, 3889316089, 354
Approximated value: 9.1097722286464436550011371408814 reached after 19 iterations

Golden ratio to 256 decimal places:
Approximated value: 1.6180339887498948482045868343656381177203091798057628621354486227052604628189024497
Reached after 616 iterations
```

Metaprogramming

Sidef recognizes all Unicode mathematical operators and allows the user to define methods that behave like infix operators, even for built-in types.

```
class Number {
    method ⊕(arg) {
        self + arg
    }
}

say (21 ⊕ 42)
```

Another example of metaprogramming, is the definition of methods at run-time:

```
var colors = Hash(
               'black'   => "000",
               'red'     => "f00",
               'green'   => "0f0",
               'yellow'  => "ff0",
               'blue'    => "00f",
               'magenta' => "f0f",
               'cyan'    => "0ff",
               'white'   => "fff",
             )

for color,code in colors {
    String.def_method("in_#{color}", func (self) {
        '<span style="color: #' + code + '">' + self + '</span>'
    })
}

say "blue".in_blue
say "red".in_red
say "white".in_white
```

Output:

```
<span style="color: #00f">blue</span>
<span style="color: #f00">red</span>
<span style="color: #fff">white</span>
```

Metronome

```
func metronome (beats_per_minute = 72, beats_per_bar = 4) {

    var counter   = 0
    var duration  = 60/beats_per_minute
    var base_time = Time.micro+duration

    STDOUT.autoflush(true)

    for next_time in (base_time..Inf `by` duration) {
        if (counter++ %% beats_per_bar) {
            print "\nTICK"
        }
        else {
            print " tick"
        }
        Sys.sleep(next_time - Time.micro)
    }
}

say metronome(ARGV.map{ Num(_) }...)
```

Output:

```
% sidef metronome.sf 60 6

TICK tick tick tick tick tick
TICK tick tick tick tick tick
TICK tick tick tick tick tick
TICK tick tick tick tick tick
TICK tick tick tick tick tick
TICK tick tick tick tick tick
TICK tick tick tick^C
```

Mian-Chowla sequence

```
var (n, sums, ts, mc) = (100, Set(2), [], [1])
var st = Time.micro
for i in (1 ..^ n) {
   for j in (mc[i-1]+1 .. Inf) {
      mc[i] = j
      for k in (0 .. i) {
         var sum = mc[k]+j
         if (sums.has(sum)) {
            ts.clear
            break
         }
         ts << sum
      }
      if (ts.len > 0) {
         sums |= Set(ts...)
         break
      }
   }
}
var et = (Time.micro - st)
var s = " of the Mian-Chowla sequence are:\n"
say "The first 30 terms#{s}#{mc.ft(0, 29).join(' ')}\n"
say "Terms 91 to 100#{s}#{mc.ft(90, 99).join(' ')}\n"
say "Computation time was #{et} seconds."
```

Output:

```
The first 30 terms of the Mian-Chowla sequence are:
1 2 4 8 13 21 31 45 66 81 97 123 148 182 204 252 290 361 401 475 565 593 662 775 822 916 970 1016 1159

Terms 91 to 100 of the Mian-Chowla sequence are:
22526 23291 23564 23881 24596 24768 25631 26037 26255 27219

Computation time was 3.41664 seconds.
```

Middle three digits

```
func middle_three(n) {
  var l = n.len
  if (l < 3) {
    "#{n} is too short"
  } elsif (l.is_even) {
    "#{n} has an even number of digits"
  } else {
    "The three middle digits of #{n} are: " + n.digits.ft(l-3 / 2, l/2 + 1).flip.join
  }
}

var nums = %n(
    123 12345 1234567 987654321 10001 -10001 -123 -100 100 -12345
    1 2 -1 -10 2002 -2002 0
)
nums.each { say middle_three(_) }
```

Output:

```
The three middle digits of 123 are: 123
The three middle digits of 12345 are: 234
The three middle digits of 1234567 are: 345
The three middle digits of 987654321 are: 654
The three middle digits of 10001 are: 000
The three middle digits of -10001 are: 000
The three middle digits of -123 are: 123
The three middle digits of -100 are: 100
The three middle digits of 100 are: 100
The three middle digits of -12345 are: 234
1 is too short
2 is too short
-1 is too short
-10 is too short
2002 has an even number of digits
-2002 has an even number of digits
0 is too short
```

Miller–Rabin primality test

```
func miller_rabin(n, k=10) {

    return false if (n <= 1)
    return true  if (n == 2)
    return false if (n.is_even)

    var t = n-1
    var s = t.valuation(2)
    var d = t>>s

    k.times {
        var a = irand(2, t)
        var x = powmod(a, d, n)
        next if (x ~~ [1, t])

        (s-1).times {
            x.powmod!(2, n)
            return false if (x == 1)
            break if (x == t)
        }

        return false if (x != t)
    }

    return true
}

say miller_rabin.grep(^1000).join(', ')
```

Minimum multiple of m where digital sum equals m

```
var e = Enumerator({|f|
    for n in (1..Inf) {

        var k = 0
        while (k.sumdigits != n) {
            k += n
        }

        f(k/n)
    }
})

var N = 60
var A = []

e.each {|v|
    A << v
    say A.splice.map { '%7s' % _ }.join(' ') if (A.len == 10)
    break if (--N <= 0)
}
```

Output:

```
     1       1       1       1       1       1       1       1       1      19
    19       4      19      19      13      28      28      11      46     199
    19     109      73      37     199      73      37     271     172    1333
   289     559    1303     847    1657     833    1027    1576    1282   17497
  4339    2119    2323   10909   11111   12826   14617   14581   16102  199999
 17449   38269   56413   37037 1108909  142498  103507  154981  150661 1333333
```

Minimum numbers of three lists

```
var lists = [
    [ 5, 45, 23, 21, 67],
    [43, 22, 78, 46, 38],
    [ 9, 98, 12, 98, 53],
]

say lists.zip.map{.min}
```

Output:

```
[5, 22, 12, 21, 38]
```

Minimum positive multiple in base 10 using only 0 and 1

Based on the Sage code by Eric M. Schmidt, which in turn is based on C code by Rick Heylen.

```
func find_B10(n, b=10) {

    return 0 if (n == 0)

    var P = n.of(-1)
    for (var m = 0; P[0] == -1; ++m) {

        for r in (0..n) {

            next if (P[r] == -1)
            next if (P[r] ==  m)

            with ((powmod(b, m, n) + r) % n) { |t|
                P[t] = m if (P[t] == -1)
            }
        }
    }

    var R = 0
    var r = 0

    do {
        R += b**P[r]
        r = (r - powmod(b, P[r], n))%n
    } while (r > 0)

    return R
}

printf("%5s: %28s  %s\n", 'Number', 'B10', 'Multiplier')

for n in (1..10, 95..105, 297, 576, 594, 891, 909, 999, 1998, 2079, 2251, 2277, 2439, 2997, 4878) {
    printf("%6d: %28s  %s\n", n, var a = find_B10(n), a/n)
}
```

Output:

```
Number:                          B10  Multiplier
     1:                            1  1
     2:                           10  5
     3:                          111  37
     4:                          100  25
     5:                           10  2
     6:                         1110  185
     7:                         1001  143
     8:                         1000  125
     9:                    111111111  12345679
    10:                           10  1
    95:                       110010  1158
    96:                     11100000  115625
    97:                     11100001  114433
    98:                     11000010  112245
    99:           111111111111111111  1122334455667789
   100:                          100  1
   101:                          101  1
   102:                      1000110  9805
   103:                     11100001  107767
   104:                      1001000  9625
   105:                       101010  962
   297:          1111011111111111111  3740778151889263
   576:              111111111000000  192901234375
   594:         11110111111111111110  18703890759446315
   891:          1111111111111111011  1247038284075321
   909:          1011111111111111111  1112333455567779
   999:  111111111111111111111111111  111222333444555666777889
  1998: 1111111111111111111111111110  556111667222778333889445
  2079:       1001101101111111111111  481530111164555609
  2251:                 101101101111  44913861
  2277:         11110111111111111011  4879275850290343
  2439:   1000010101111101111011111111  4100082415379299344449
  2997: 1111110111111111111111111111111  370740777814851888925963
  4878:  100001010111101111011111110  20500412076896496722245
```

Minimum primes

```
var lists = [
    [ 5,45,23,21,67],
    [43,22,78,46,38],
    [ 9,98,12,54,53],
]

say lists.zip.map { next_prime(.max - 1) }
```

Output:

```
[43, 101, 79, 59, 67]
```

Modular arithmetic

```
class Modulo(n=0, m=13) {

  method init {
     (n, m) = (n % m, m)
  }

  method to_n { n }

  < + - * ** >.each { |meth|
      Modulo.def_method(meth, method(n2) { Modulo(n.(meth)(n2.to_n), m) })
  }

  method to_s { "#{n} ⌈mod #{m}⌋" }
}

func f(x) { x**100 + x + 1 }
say f(Modulo(10, 13))
```

Output:

```
1 ⌈mod 13⌋
```

Modular exponentiation

Built-in:

```
say expmod(
    29883481620585741369158914214988194663201633129269524237910230788761 39,
    23513993033734644864661225445236900947449752334155440729926568812403 19,
    10**40)
```

User-defined:

```
func expmod(a, b, n) {
    var c = 1
    do {
        (c *= a) %= n if b.is_odd
        (a *= a) %= n
    } while (b //= 2)
    c
}
```

Output:

```
152722999858524845001680895834374045 3059
```

Modular inverse

Built-in:

```
say 42.modinv(2017)
```

Algorithm implementation:

```
func invmod(a, n) {
  var (t, nt, r, nr) = (0, 1, n, a % n)
  while (nr != 0) {
    var quot = int((r - (r % nr)) / nr);
    (nt, t) = (t - quot*nt, nt);
    (nr, r) = (r - quot*nr, nr);
  }
  r > 1 && return()
  t < 0 && (t += n)
  t
}

say invmod(42, 2017)
```

Output:

```
1969
```

Modulinos

```
# Life.sm

func meaning_of_life {
    42
}

if (__FILE__ == __MAIN__) {
    say "Main: The meaning of life is #{meaning_of_life()}"
}
```

```
# test.sf

include Life

say "Test: The meaning of life is #{Life::meaning_of_life()}."
```

Monte Carlo methods

```
func monteCarloPi(nthrows) {
    4 * (^nthrows -> count_by {
        hypot(1.rand(2) - 1, 1.rand(2) - 1) < 1
    }) / nthrows
}

for n in [1e2, 1e3, 1e4, 1e5, 1e6] {
    printf("%9d: %07f\n", n, monteCarloPi(n))
}
```

Output:

```
     100: 3.320000
    1000: 3.120000
   10000: 3.169600
  100000: 3.138920
 1000000: 3.142344
```

Montgomery reduction

```
func montgomeryReduce(m, a) {
    {
        a += m if a.is_odd
        a >>= 1
    } * m.as_bin.len

    a % m
}

var m  = 75079109464472655964063840769 9
var t1 = 323165824550862327179367294465482435542970161392400401329100

var r1 = 440160025148131680164261562101
var r2 = 435362628198191204145287283255

var x1 = 540019781128412936473322405310
var x2 = 515692107665463680305819378593

say("Original x1:        ", x1)
say("Recovererd from r1: ", montgomeryReduce(m, r1))
say("Original x2:        ", x2)
say("Recovererd from r2: ", montgomeryReduce(m, r2))

print("\nMontgomery computation of x1^x2 mod m:    ")
var prod = montgomeryReduce(m, t1/x1)
var base = montgomeryReduce(m, t1)

for (var exponent = x2; exponent ; exponent >>= 1) {
    prod = montgomeryReduce(m, prod * base) if exponent.is_odd
    base = montgomeryReduce(m, base * base)
}

say(montgomeryReduce(m, prod))
say("Library-based computation of x1^x2 mod m: ", x1.powmod(x2, m))
```

Output:

```
Original x1:        540019781128412936473322405310
Recovererd from r1: 540019781128412936473322405310
Original x2:        515692107665463680305819378593
Recovererd from r2: 515692107665463680305819378593

Montgomery computation of x1^x2 mod m:    151232511393500655853002423778
Library-based computation of x1^x2 mod m: 151232511393500655853002423778
```

Monty Hall problem

```
var n = 1000                                    # number of times to play
var switchWins = (var stayWins = 0)             # sum of each strategy's wins

n.times {                                       # play the game n times
   var prize = pick(^3)
   var chosen = pick(^3)

   var show;
   do {
        show = pick(^3)
   } while (show ~~ [chosen, prize])

   given(chosen) {
     when (prize)                 { stayWins   += 1 }
     when ([3 - show - prize])    { switchWins += 1 }
     default                      { die "~ error ~" }
   }
}

say ("Staying wins %.2f%% of the time."   % (100.0 * stayWins   / n))
say ("Switching wins %.2f%% of the time." % (100.0 * switchWins / n))
```

Output:

```
Staying wins 30.10% of the time.
Switching wins 69.90% of the time.
```

Mosaic matrix

```
func mosaic_matrix(n) {
    n.of {|k|
        var(a,b) = (k.is_even ? (1,0) : (0,1))
        n.of {|j| j.is_even ? a : b }
    }
}

mosaic_matrix(5).each { .join(' ').say }; say ''
mosaic_matrix(6).each { .join(' ').say }
```

Output:

```
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1

1 0 1 0 1 0
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1
```

Most frequent k chars distance

```
func _MostFreqKHashing(string, k) {

    var seen = Hash()
    var chars = string.chars
    var freq = chars.freq
    var schars = freq.keys.sort_by {|c| -freq{c} }

    var mfkh = []
    for i in ^k {
        chars.each { |c|
            seen{c} && next
            if (freq{c} == freq{schars[i]}) {
                seen{c} = true
                mfkh << Hash(c => c, f => freq{c})
                break
            }
        }
    }

    mfkh << (k-seen.len -> of { Hash(c => :NULL, f => 0) }...)
    mfkh
}

func MostFreqKSDF(a, b, k, d) {

    var mfkh_a = _MostFreqKHashing(a, k);
    var mfkh_b = _MostFreqKHashing(b, k);

    d - gather {
        mfkh_a.each { |s|
            s{:c} == :NULL && next
            mfkh_b.each { |t|
                s{:c} == t{:c} &&
                    take(s{:f} + (s{:f} == t{:f} ? 0 : t{:f}))
            }
        }
    }.sum
}

func MostFreqKHashing(string, k) {
    gather {
        _MostFreqKHashing(string, k).each { |h|
            take("%s%d" % (h{:c}, h{:f}))
        }
    }.join
}


var str1 = "LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV"
var str2 = "EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG"

say "str1 = #{str1.dump}"
say "str2 = #{str2.dump}"

say ''

say("MostFreqKHashing(str1, 2) = ", MostFreqKHashing(str1, 2))
say("MostFreqKHashing(str2, 2) = ", MostFreqKHashing(str2, 2))
say("MostFreqKSDF(str1, str2, 2, 100) = ", MostFreqKSDF(str1, str2, 2, 100))

say ''

var arr = [
    %w(night nacht)
```

```
    %w(night nacht),
    %w(my a),
    %w(research research),
    %w(aaaaabbbb abababaaa),
    %w(significant capabilities),
]

var k = 2
var limit = 10

for s,t in arr {
    "mfkh(%s, %s, #{k}) = [%s, %s] (SDF: %d)\n".printf(
        s.dump, t.dump,
        MostFreqKHashing(s, k).dump,
        MostFreqKHashing(t, k).dump,
        MostFreqKSDF(s, t, k, limit),
    )
}
```

Output:

```
str1 = "LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV"
str2 = "EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG"

MostFreqKHashing(str1, 2) = L9T8
MostFreqKHashing(str2, 2) = F9L8
MostFreqKSDF(str1, str2, 2, 100) = 83

mfkh("night", "nacht", 2) = ["n1i1", "n1a1"] (SDF: 9)
mfkh("my", "a", 2) = ["m1y1", "a1NULL0"] (SDF: 10)
mfkh("research", "research", 2) = ["r2e2", "r2e2"] (SDF: 6)
mfkh("aaaaabbbb", "abababaaa", 2) = ["a5b4", "a5b4"] (SDF: 1)
mfkh("significant", "capabilities", 2) = ["i3n2", "i3a2"] (SDF: 7)
```

Motzkin numbers

Built-in:

```
say 50.of { .motzkin }
```

Motzkin's triangle (the Motzkin numbers are on the hypotenuse of the triangle):

```
func motzkin_triangle(num, callback) {
    var row = []
    { |n|
        row = [0, 1, {|i| row[i+2] + row[i] + row[i+1] }.map(0 .. n-3)..., 0]
        callback(row.grep{ .> 0 })
    } << 2..(num+1)
}

motzkin_triangle(10, {|row|
    row.map { "%4s" % _ }.join(' ').say
})
```

Output:

```
   1
   1    1
   1    2    2
   1    3    5    4
   1    4    9   12    9
   1    5   14   25   30   21
   1    6   20   44   69   76   51
   1    7   27   70  133  189  196  127
   1    8   35  104  230  392  518  512  323
   1    9   44  147  369  726 1140 1422 1353  835
```

Task:

```
func motzkin_numbers(N) {

    var (a, b, n) = (0, 1, 1)

    N.of {
        var M = b/n
        n += 1
        (a, b) = (b, (3*(n-1)*n*a + (2*n - 1)*n*b) / ((n+1)*(n-1)))
        M
    }
}

motzkin_numbers(42).each_kv {|k,v|
    say "#{'%2d' % k}: #{v}#{v.is_prime ? ' prime' : ''}"
}
```

Output:

```
 0: 1
 1: 1
 2: 2 prime
 3: 4
 4: 9
 5: 21
 6: 51
 7: 127 prime
 8: 323
 9: 835
10: 2188
11: 5798
12: 15511 prime
13: 41835
14: 113634
15: 310572
16: 853467
17: 2356779
18: 6536382
19: 18199284
20: 50852019
21: 142547559
22: 400763223
23: 1129760415
24: 3192727797
25: 9043402501
26: 25669818476
27: 73007772802
28: 208023278209
29: 593742784829
30: 1697385471211
31: 4859761676391
32: 13933569346707
33: 40002464776083
34: 114988706524270
35: 330931069469828
36: 953467954114363 prime
37: 2750016719520991
38: 7939655757745265
39: 22944749046030949
40: 66368199913921497
41: 192137918101841817
```

Move-to-front algorithm

Implemented using regular expressions:

```
func encode(str) {
    var table = ('a'..'z' -> join)
    str.chars.map { |c|
        var s = ''
        table.sub!(Regex('(.*?)' + c), {|s1| s=s1; c + s1})
        s.len
    }
}

func decode(nums) {
    var table = ('a'..'z' -> join)
    nums.map { |n|
        var s = ''
        table.sub!(Regex('(.{' + n + '})(.)'), {|s1, s2| s=s2; s2 + s1})
        s
    }.join
}

%w(broood bananaaa hiphophiphop).each { |test|
    var encoded = encode(test)
    say "#{test}: #{encoded}"
    var decoded = decode(encoded)
    print "in" if (decoded != test)
    say "correctly decoded to #{decoded}"
}
```

Alternatively, implemented as a module, using arrays:

```
module MoveToFront {

  define ABC = @("a".."z")

  func m2f(ar,i) {
    [ar.delete_index(i)] + ar
  }

  func encode(str) {
    var ar = ABC+[]
    gather {
      str.each_char { |char|
        take(var i = ar.index(char))
        ar = m2f(ar, i)
      }
    }
  }

  func decode(indices) {
    var ar = ABC+[]
    gather {
      indices.each { |i|
        take ar[i]
        ar = m2f(ar, i)
      }
    }.join
  }
}

%w(broood bananaaa hiphophiphop).each { |test|
    var encoded = MoveToFront::encode(test)
    say "#{test}: #{encoded}"
    var decoded = MoveToFront::decode(encoded)
    print "in" if (decoded != test)
    say "correctly decoded to #{decoded}"
}
```

Output:

```
broood: 1 17 15 0 0 5
correctly decoded to broood
bananaaa: 1 1 13 1 1 1 0 0
correctly decoded to bananaaa
hiphophiphop: 7 8 15 2 15 2 2 3 2 2 3 2
correctly decoded to hiphophiphop
```

Multi-base primes

```
func max_prime_bases(ndig, maxbase=36) {

    var maxprimebases = [[]]
    var nwithbases = [0]
    var maxprime = (10**ndig - 1)

    for p in (idiv(maxprime + 1, 10) .. maxprime) {
        var dig = p.digits
        var bases = (2..maxbase -> grep {|b| dig.all { _ < b } && dig.digits2num(b).is_prime })
        if (bases.len > maxprimebases.first.len) {
            maxprimebases = [bases]
            nwithbases = [p]
        }
        elsif (bases.len == maxprimebases.first.len) {
            maxprimebases << bases
            nwithbases << p
        }
    }

    var (alen, vlen) = (maxprimebases.first.len, maxprimebases.len)

    say("\nThe maximum number of prime valued bases for base 10 numeric strings of length ",
        ndig, " is #{alen}. The base 10 value list of ", vlen > 1 ? "these" : "this", " is:")
    maxprimebases.each_kv {|k,v| say(nwithbases[k], " => ", v) }
}

for n in (1..5) {
    max_prime_bases(n)
}
```

Output:

```
The maximum number of prime valued bases for base 10 numeric strings of length 1 is 34. The base 10 val
2 => [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,

The maximum number of prime valued bases for base 10 numeric strings of length 2 is 18. The base 10 val
21 => [3, 5, 6, 8, 9, 11, 14, 15, 18, 20, 21, 23, 26, 29, 30, 33, 35, 36]

The maximum number of prime valued bases for base 10 numeric strings of length 3 is 18. The base 10 val
131 => [4, 5, 7, 8, 9, 10, 12, 14, 15, 18, 19, 20, 23, 25, 27, 29, 30, 34]
551 => [6, 7, 11, 13, 14, 15, 16, 17, 19, 21, 22, 24, 25, 26, 30, 32, 35, 36]
737 => [8, 9, 11, 12, 13, 15, 16, 17, 19, 22, 23, 24, 25, 26, 29, 30, 31, 36]

The maximum number of prime valued bases for base 10 numeric strings of length 4 is 19. The base 10 val
1727 => [8, 9, 11, 12, 13, 15, 16, 17, 19, 20, 22, 23, 24, 26, 27, 29, 31, 33, 36]
5347 => [8, 9, 10, 11, 12, 13, 16, 18, 19, 22, 24, 25, 26, 30, 31, 32, 33, 34, 36]

The maximum number of prime valued bases for base 10 numeric strings of length 5 is 18. The base 10 val
30271 => [8, 10, 12, 13, 16, 17, 18, 20, 21, 23, 24, 25, 31, 32, 33, 34, 35, 36]
```

Multifactorial

```
func mfact(s, n) {
    n > 0 ? (n * mfact(s, n-s)) : 1
}

{ |s|
    say "step=#{s}: #{{|n| mfact(s, n)}.map(1..10).join(' ')}"
} << 1..10
```

Output:

```
step=1: 1 2 6 24 120 720 5040 40320 362880 3628800
step=2: 1 2 3 8 15 48 105 384 945 3840
step=3: 1 2 3 4 10 18 28 80 162 280
step=4: 1 2 3 4 5 12 21 32 45 120
step=5: 1 2 3 4 5 6 14 24 36 50
step=6: 1 2 3 4 5 6 7 16 27 40
step=7: 1 2 3 4 5 6 7 8 18 30
step=8: 1 2 3 4 5 6 7 8 9 20
step=9: 1 2 3 4 5 6 7 8 9 10
step=10: 1 2 3 4 5 6 7 8 9 10
```

Multiline shebang

```
#!/bin/sh

#`(if running under some shell) {
    eval 'exec /usr/bin/sidef $0 ${1+"$@"} "world"'
}

say "Hello, #{ARGV[0]}!"
```

Output:

```
$ ./script.sf
Hello, world!

$ ./script.sf Sidef
Hello, Sidef!

$ sidef script.sf RosettaCode
Hello, RosettaCode!
```

Multiple distinct objects

```
[Foo.new] * n       # incorrect (only one distinct object is created)
```

```
n.of {Foo.new}      # correct
```

Multiplication tables

```
var max = 12
var width = (max**2 -> len+1)

func fmt_row(*items) {
    items.map {|s| "%*s" % (width, s) }.join
}

say fmt_row('x│', (1..max)...)
say "#{'─' * (width - 1)}┼#{'─' * (max * width)}"

{ |i|
    say fmt_row("#{i}│", {|j| i <= j ? i*j : ''}.map(1..max)...)
} << 1..max
```

Output:

```
  x│   1   2   3   4   5   6   7   8   9  10  11  12
───┼────────────────────────────────────────────────
  1│   1   2   3   4   5   6   7   8   9  10  11  12
  2│       4   6   8  10  12  14  16  18  20  22  24
  3│           9  12  15  18  21  24  27  30  33  36
  4│              16  20  24  28  32  36  40  44  48
  5│                  25  30  35  40  45  50  55  60
  6│                      36  42  48  54  60  66  72
  7│                          49  56  63  70  77  84
  8│                              64  72  80  88  96
  9│                                  81  90  99 108
 10│                                     100 110 120
 11│                                         121 132
 12│                                             144
```

Multiplicative order

Built-in:

```
say 37.znorder(1000)     #=> 100
say 54.znorder(100001)   #=> 9090
```

User-defined:

```
func mo_prime(a, p, e) {
    var m  = p**e
    var t  = (p-1)*(p**(e-1))
    var qs = [1]

    for p,e in (t.factor_exp) {
        qs.map! {|q|
            0..e -> map {|j| q * p**j }...
        }
    }

    qs.sort.first_by {|q| powmod(a, q, m) == 1 }
}

func mo(a, m) {
    gcd(a, m) == 1 || die "#{a} and #{m} are not relatively prime"
    Math.lcm(1, m.factor_exp.map {|r| mo_prime(a, r...) }...)
}

say mo(37, 1000)
say mo(54, 100001)

with (10**20 - 1) {|b|
    say mo(2, b)
    say mo(17, b)
}
```

Output:

```
100
9090
3748806900
1499522760
```

Multisplit

```
func multisplit(sep, str, keep_sep=false) {
    sep = sep.map{.escape}.join('|')
    var re = Regex(keep_sep ? "(#{sep})" : sep)
    str.split(re, -1)
}

[false, true].each { |bool|
    say multisplit(%w(== != =), 'a!===b=!=c', keep_sep: bool)
}
```

Output:

```
["a", "", "b", "", "c"]
["a", "!=", "", "==", "b", "=", "", "!=", "c"]
```

Munchausen numbers

```
func is_munchausen(n) {
    n.digits.map{|d| d**d }.sum == n
}

say (1..5000 -> grep(is_munchausen))
```

Output:

```
[1, 3435]
```

Munching squares

```
require('Imager')

var img = %O<Imager>.new(xsize => 256, ysize => 256)

for y=^256, x=^256 {
    var rgb = [(255 - x - y).abs, (255-x)^y, x^(255-y)]
    img.setpixel(x => x, y => y, color => rgb)
}

img.write(file => 'xor.png')
```

Output image

Mutual recursion

```
func F(){}
func M(){}

F = func(n) { n > 0 ? (n - M(F(n-1))) : 1 }
M = func(n) { n > 0 ? (n - F(M(n-1))) : 0 }

[F, M].each { |seq|
    {|i| seq.call(i)}.map(^20).join(' ').say
}
```

Output:

```
1 1 2 2 3 3 4 5 5 6 6 7 8 8 9 9 10 11 11 12
0 0 1 2 2 3 4 4 5 6 6 7 7 8 9 9 10 11 11 12
```

N'th

```
func nth(n) {
    static irregulars = Hash(<1 st 2 nd 3 rd 11 th 12 th 13 th>...)
    n.to_s + (irregulars{n % 100} \\ irregulars{n % 10} \\ 'th')
}

for r in [0..25, 250..265, 1000..1025] {
    say r.map {|n| nth(n) }.join(" ")
}
```

Output:

```
0th 1st 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th 17th 18th 19th 20th 21st 22n
250th 251st 252nd 253rd 254th 255th 256th 257th 258th 259th 260th 261st 262nd 263rd 264th 265th
1000th 1001st 1002nd 1003rd 1004th 1005th 1006th 1007th 1008th 1009th 1010th 1011th 1012th 1013th 1014t
```

N-queens problem

```
func N_queens_solution(N = 8) {

    func collision(field, row) {
        for i in (^row) {
            var distance = (field[i] - field[row])
            distance ~~ [0, row-i, i-row] && return true
        }
        return false
    }

    func search(field, row) {
        row == N && return field
        for i in (^N) {
            field[row] = i
            if (!collision(field, row)) {
                return (__FUNC__(field, row+1) || next)
            }
        }
        return []
    }

    for i in (0 .. N>>1) {
        if (var r = search([i], 1)) {
            return r
        }
    }
}

for n in (1..15) {
    say "#{'%2d' % n}: #{N_queens_solution(n) || 'No solution'}"
}
```

Output:

```
 1: [0]
 2: No solution
 3: No solution
 4: [1, 3, 0, 2]
 5: [0, 2, 4, 1, 3]
 6: [1, 3, 5, 0, 2, 4]
 7: [0, 2, 4, 6, 1, 3, 5]
 8: [0, 4, 7, 5, 2, 6, 1, 3]
 9: [0, 2, 5, 7, 1, 3, 8, 6, 4]
10: [0, 2, 5, 7, 9, 4, 8, 1, 3, 6]
11: [0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
12: [0, 2, 4, 7, 9, 11, 5, 10, 1, 6, 8, 3]
13: [0, 2, 4, 1, 8, 11, 9, 12, 3, 5, 7, 10, 6]
14: [0, 2, 4, 6, 11, 9, 12, 3, 13, 8, 1, 5, 7, 10]
15: [0, 2, 4, 1, 9, 11, 13, 3, 12, 8, 5, 14, 6, 10, 7]
```

N-smooth numbers

```
func smooth_generator(primes) {
    var s = primes.len.of { [1] }
    {
        var n = s.map { .first }.min
        { |i|
            s[i].shift if (s[i][0] == n)
            s[i] << (n * primes[i])
        } * primes.len
        n
    }
}

for p in (primes(2,29)) {
    var g = smooth_generator(p.primes)
    say ("First 25 #{'%2d'%p}-smooth numbers: ", 25.of { g.run }.join(' '))
}

say ''

for p in (primes(3,29)) {
    var g = smooth_generator(p.primes)
    say ("3,000th through 3,002nd #{'%2d'%p}-smooth numbers: ", 3002.of { g.run }.last(3).join(' '))
}
```

Output:

```
First 25  2-smooth numbers: 1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 2
First 25  3-smooth numbers: 1 2 3 4 6 8 9 12 16 18 24 27 32 36 48 54 64 72 81 96 108 128 144 162 192
First 25  5-smooth numbers: 1 2 3 4 5 6 8 9 10 12 15 16 18 20 24 25 27 30 32 36 40 45 48 50 54
First 25  7-smooth numbers: 1 2 3 4 5 6 7 8 9 10 12 14 15 16 18 20 21 24 25 27 28 30 32 35 36
First 25 11-smooth numbers: 1 2 3 4 5 6 7 8 9 10 11 12 14 15 16 18 20 21 22 24 25 27 28 30 32
First 25 13-smooth numbers: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 24 25 26 27 28
First 25 17-smooth numbers: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 21 22 24 25 26 27
First 25 19-smooth numbers: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 24 25 26
First 25 23-smooth numbers: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
First 25 29-smooth numbers: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

3,000th through 3,002nd  3-smooth numbers: 915803679783062524417246494 72 928298231864148199155475415 04
3,000th through 3,002nd  5-smooth numbers: 278942752080 279936000000 281250000000
3,000th through 3,002nd  7-smooth numbers: 50176000 50331648 50388480
3,000th through 3,002nd 11-smooth numbers: 2112880 2116800 2117016
3,000th through 3,002nd 13-smooth numbers: 390000 390390 390625
3,000th through 3,002nd 17-smooth numbers: 145800 145860 146016
3,000th through 3,002nd 19-smooth numbers: 74256 74358 74360
3,000th through 3,002nd 23-smooth numbers: 46552 46575 46585
3,000th through 3,002nd 29-smooth numbers: 33516 33524 33534
```

Optionally, an efficient algorithm for checking if a given arbitrary large number is smooth over a given product of primes:

```
func is_smooth_over_prod(n, k) {

    return true  if (n == 1)
    return false if (n <= 0)

    for (var g = gcd(n,k); g > 1; g = gcd(n,k)) {
        n /= g**valuation(n,g)        # remove any divisibility by g
        return true if (n == 1)       # smooth if n == 1
    }

    return false
}

for p in (503, 509, 521) {
    var k = p.primorial
    var a = {|n| is_smooth_over_prod(n, k) }.first(30_019).last(20)
    say ("30,000th through 30,019th #{p}-smooth numbers: ", a.join(' '))
}
```

Output:

```
30,000th through 30,019th 503-smooth numbers: 62913 62914 62916 62918 62920 62923 62926 62928 62930 629
30,000th through 30,019th 509-smooth numbers: 62601 62602 62604 62607 62608 62609 62611 62618 62620 626
30,000th through 30,019th 521-smooth numbers: 62287 62288 62291 62292 62300 62304 62307 62308 62310 623
```

Named parameters

```
func example(foo: 0, bar: 1, grill: "pork chops") {
    say "foo is #{foo}, bar is #{bar}, and grill is #{grill}";
}

# Note that :foo is omitted and :grill precedes :bar
example(grill: "lamb kebab", bar: 3.14);
```

Output:

```
foo is 0, bar is 3.14, and grill is lamb kebab
```

Names to numbers

```
func names_to_number(str) {

    static nums = Hash.new(
           zero => 0,             one => 1,             two => 2,
          three => 3,            four => 4,            five => 5,
            six => 6,           seven => 7,           eight => 8,
           nine => 9,             ten => 10,          eleven => 11,
         twelve => 12,       thirteen => 13,       fourteen => 14,
       fifteen => 15,        sixteen => 16,      seventeen => 17,
      eighteen => 18,       nineteen => 19,         twenty => 20,
        thirty => 30,          forty => 40,          fifty => 50,
         sixty => 60,        seventy => 70,         eighty => 80,
        ninety => 90,        hundred => 1e2,      thousand => 1e3,
       million => 1e6,       billion => 1e9,      trillion => 1e12,
    quadrillion => 1e15,  quintillion => 1e18,
    );

    # Groupings for thousands, millions, ..., quintillions
    static groups = /\d{4}|\d{7}|\d{10}|\d{13}|\d{16}|\d{19}/;

    # Numeral
    static num = /\d+/;

    str.trim!;                      # remove leading and trailing whitespace
    str.gsub!('-', ' ');            # convert hyphens to spaces
    str.words!.join!(' ');          # remove duplicate whitespace, convert ws to space
    str.lc!;                        # convert to lower case

    # tokenize sentence boundaries
    str.gsub!(/([.?!]) /, {|a| ' ' + a + "\n"});
    str.gsub!(/([.?!])$/, {|a| ' ' + a + "\n"});

    # tokenize other punctuation and symbols
    str.gsub!(/\$(.)/,           {|a| "$ #{a}" });       # prefix
    str.gsub!(/(.)([;:%'',])/, {|a,b| "#{a} #{b}"});     # suffix

    nums.each { |key, value| str.gsub!(Regex.new('\b' + key + '\b'), value) };

    str.gsub!(/(\d) , (\d)/,   {|a,b| a + ' ' + b});
    str.gsub!(/(\d) and (\d)/, {|a,b| a + ' ' + b});

    static regex = [
        Regex.new('\b(\d) 100 (\d\d) (\d) (' + groups + ')\b'),
        Regex.new('\b(\d) 100 (\d\d) (' + groups + ')\b'),
        Regex.new('\b(\d) 100 (\d) (' + groups + ')\b'),
        Regex.new('\b(\d) 100 (' + groups + ')\b'),
```

```
        Regex.new('\b100 (\d\d) (\d) (' + groups + ')\b'),
        Regex.new('\b100 (\d\d) (' + groups + ')\b'),
        Regex.new('\b100 (\d) (' + groups + ')\b'),
        Regex.new('\b100 (' + groups + ')\b'),
        Regex.new('\b(\d\d) (\d) (' + groups + ')\b'),
        Regex.new('\b(\d{1,2}) (' + groups + ')\b'),
        Regex.new('((?:' + num + ' )*' + num + ')'),
    ];

    str.gsub!(regex[0], {|a,b,c,d| (a.to_i*100 + b.to_i + c.to_i) * d.to_i });
    str.gsub!(regex[1], {|a,b,c|   (a.to_i*100 + b.to_i) * c.to_i });
    str.gsub!(regex[2], {|a,b,c|   (a.to_i*100 + b.to_i) * c.to_i });
    str.gsub!(regex[3], {|a,b|     (a.to_i * b.to_i * 100) });
    str.gsub!(regex[4], {|a,b,c|   (100 + a.to_i + b.to_i) * c.to_i });
    str.gsub!(regex[5], {|a,b|     (100 + a.to_i) * b.to_i });
    str.gsub!(regex[6], {|a,b|     (100 + a.to_i) * b.to_i });
    str.gsub!(regex[7], {|a|       (a.to_i * 100) });
    str.gsub!(regex[8], {|a,b,c|   (a.to_i + b.to_i) * c.to_i });
    str.gsub!(regex[9], {|a,b|     (a.to_i * b.to_i) });

    str.gsub!(/\b(\d\d) (\d) 100\b/, {|a,b| (a.to_i + b.to_i) * 100});
    str.gsub!(/\b(\d{1,2}) 100\b/,   {|a|   (a.to_i * 100) });
    str.gsub!(/\b(\d{2}) (\d{2})\b/, {|a,b| (a.to_i * 100) + b.to_i});
    str.gsub!(regex[10], {|a| a.split(' ').map{.to_i}.sum });
}
```

Usage:

```
ARGF.each { |line|
    say "#{line.chomp.dump} --> #{names_to_number(line).dump}";
}
```

Sample:

Output:

```
$ sidef names2nums.sf input.txt
"Seventy two dollars" --> "72 dollars"
"One Hundred and One Dalmatians" --> "101 dalmatians"
"Two Thousand and One: A Space Odyssey" --> "2001 : a space odyssey"
"Twenty Thirteen" --> "2013"
"Nineteen Eighty-Four" --> "1984"
"one thousand, five hundred and one" --> "1501"
"three hundred and ten" --> "310"
"ninety-nine" --> "99"
"ninety nine thousand nine hundred ninety nine" --> "99999"
"five hundred and twelve thousand, six hundred and nine" --> "512609"
"two billion, one hundred" --> "2000000100"
```

Narcissist

```
say (File(__FILE__).open_r.slurp == ARGF.slurp)
```

Narcissistic decimal number

```
func is_narcissistic(n) {
    n.digits »**» n.len -> sum == n
}

var count = 0
for i in ^Inf {
    if (is_narcissistic(i)) {
        say "#{++count}\t#{i}"
        break if (count == 25)
    }
}
```

Output:

```
1       0
2       1
3       2
4       3
5       4
6       5
7       6
8       7
9       8
10      9
11      153
12      370
13      371
14      407
15      1634
16      8208
17      9474
18      54748
19      92727
20      93084
21      548834
22      1741725
23      4210818
24      9800817
25      9926315
```

Native shebang

Sidef is a scripting language and does not compile to a binary.

```
#!/usr/bin/sidef
say ARGV.join(" ")
```

Output:

```
$ ./echo.sf Hello, World!
Hello, World!
```

Natural sorting

```
class String {
    # Sort groups of digits in number order. Sort by order of magnitude then lexically.
    -> naturally { self.lc.gsub(/(\d+)/, {|s1| "0" + s1.len.chr + s1 }) + "\x0" + self };

    # Collapse multiple ws characters to a single.
    -> collapse { self.gsub(/(\s)\1+/, {|s1| s1 }) };

    # Convert all ws characters to a space.
    -> normalize { self.gsub(/(\s)/, ' ') };

    # Ignore common leading articles for title sorts
    -> title { self.sub(/^(?:a|an|the)\b\s*/i, '') };

    # Decompose ISO-Latin1 glyphs to their base character.
    -> latin1_decompose {
        static tr = Hash.new(%w(
           Æ AE æ ae Þ TH þ th Ð TH ð th ß ss À A Á A Â A Ã A Ä A Å A à a á a
            â a ã a ä a å a Ç C ç c È E É E Ê E Ë E è e é e ê e ë e Ì I Í I Î
            I Ï I ì i í i î i ï i Ò O Ó O Ô O Õ O Ö O Ø O ò o ó o ô o õ o ö o
            ø o Ñ N ñ n Ù U Ú U Û U Ü U ù u ú u û u ü u Ý Y ÿ y ý y
        )...);

        var re = Regex.new('(' + tr.keys.join('|') + ')');
        self.gsub(re, {|s1| tr{s1} });
    }
}
```

Tests:

```
var tests = [
    [
        "Task 1a\nSort while ignoring leading spaces.",
        [
          'ignore leading spaces: 1', '   ignore leading spaces: 4',
          '  ignore leading spaces: 3', ' ignore leading spaces: 2'
        ],
        { .trim } # builtin method.
    ],
    [
        "Task 1b\nSort while ignoring multiple adjacent spaces.",
        [
          'ignore m.a.s   spaces: 3', 'ignore m.a.s spaces: 1',
          'ignore m.a.s    spaces: 4', 'ignore m.a.s  spaces: 2'
        ],
        { .collapse }
    ],
    [
        "Task 2\nSort with all white space normalized to regular spaces.",
        [
          "Normalized\tspaces: 4", "Normalized\xa0spaces: 1",
          "Normalized\x20spaces: 2", "Normalized\nspaces: 3"
        ],
        { .normalize }
    ],
    [
        "Task 3\nSort case independently.",
        [
          'caSE INDEPENDENT: 3', 'casE INDEPENDENT: 2',
          'cASE INDEPENDENT: 4', 'case INDEPENDENT: 1'
        ],
        { .lc } # builtin method
    ],
    [
```

```
        "Task 4\nSort groups of digits in natural number order.",
        %w(Foo100bar99baz0.txt foo100bar10baz0.txt foo1000bar99baz10.txt
           foo1000bar99baz9.txt 201st 32nd 3rd 144th 17th 2 95),
        { .naturally }
    ],
    [
        "Task 5 ( mixed with 1, 2, 3 & 4 )\n"
        + "Sort titles, normalize white space, collapse multiple spaces to\n"
        + "single, trim leading white space, ignore common leading articles\n"
        + 'and sort digit groups in natural order.',
        [
          'The Wind     in the Willows  8', '  The 39 Steps               3',
          'The    7th Seal              1', 'Wanda                        6',
          'A Fish Called Wanda          5', ' The Wind and the Lion       7',
          'Any Which Way But Loose      4', '12 Monkeys                   2'
        ],
        { .normalize.collapse.trim.title.naturally }
    ],
    [
        "Task 6, 7, 8\nMap letters in Latin1 that have accents or decompose to two\n"
        + 'characters to their base characters for sorting.',
        %w(apple Ball bald car Card above Æon æon aether
            niño nina e-mail Évian evoke außen autumn),
        { .latin1_decompose.naturally }
    ]
];

tests.each { |case|
    var code = case.pop;
    var array = case.pop;
    say case.pop+"\n";

    say "Standard Sort:\n";
    array.sort.each { .say };

    say "\nNatural Sort:\n";
    array.sort_by(code).each { .say };

    say "\n#{'*' * 40}\n";
}
```

Output:

```
Task 1a
Sort while ignoring leading spaces.

Standard Sort:

   ignore leading spaces: 4
  ignore leading spaces: 3
 ignore leading spaces: 2
ignore leading spaces: 1

Natural Sort:

ignore leading spaces: 1
 ignore leading spaces: 2
  ignore leading spaces: 3
   ignore leading spaces: 4

****************************************

Task 1b
Sort while ignoring multiple adjacent spaces.
```

```
Standard Sort:

ignore m.a.s    spaces: 4
ignore m.a.s   spaces: 3
ignore m.a.s  spaces: 2
ignore m.a.s spaces: 1

Natural Sort:

ignore m.a.s spaces: 1
ignore m.a.s  spaces: 2
ignore m.a.s   spaces: 3
ignore m.a.s    spaces: 4

**************************************

Task 2
Sort with all white space normalized to regular spaces.

Standard Sort:

Normalized      spaces: 4
Normalized
spaces: 3
Normalized spaces: 2
Normalized spaces: 1

Natural Sort:

Normalized spaces: 1
Normalized spaces: 2
Normalized
spaces: 3
Normalized      spaces: 4

**************************************

Task 3
Sort case independently.

Standard Sort:

cASE INDEPENDENT: 4
caSE INDEPENDENT: 3
casE INDEPENDENT: 2
case INDEPENDENT: 1

Natural Sort:

case INDEPENDENT: 1
casE INDEPENDENT: 2
caSE INDEPENDENT: 3
cASE INDEPENDENT: 4

**************************************

Task 4
Sort groups of digits in natural number order.

Standard Sort:

144th
17th
2
201st
```

```
201st
32nd
3rd
95
Foo100bar99baz0.txt
foo1000bar99baz10.txt
foo1000bar99baz9.txt
foo100bar10baz0.txt

Natural Sort:

2
3rd
17th
32nd
95
144th
201st
foo100bar10baz0.txt
Foo100bar99baz0.txt
foo1000bar99baz9.txt
foo1000bar99baz10.txt

**************************************

Task 5 ( mixed with 1, 2, 3 & 4 )
Sort titles, normalize white space, collapse multiple spaces to
single, trim leading white space, ignore common leading articles
and sort digit groups in natural order.

Standard Sort:

  The 39 Steps               3
 The Wind and the Lion       7
12 Monkeys                   2
A Fish Called Wanda          5
Any Which Way But Loose      4
The    7th Seal              1
The Wind     in the Willows  8
Wanda                        6

Natural Sort:

The    7th Seal              1
12 Monkeys                   2
  The 39 Steps               3
Any Which Way But Loose      4
A Fish Called Wanda          5
Wanda                        6
 The Wind and the Lion       7
The Wind     in the Willows  8

**************************************

Task 6, 7, 8
Map letters in Latin1 that have accents or decompose to two
characters to their base characters for sorting.

Standard Sort:

Ball
Card
above
aether
apple
autumn
```

```
außen
bald
car
e-mail
evoke
nina
niño
Æon
Évian
æon

Natural Sort:

above
Æon
æon
aether
apple
außen
autumn
bald
Ball
car
Card
e-mail
Évian
evoke
nina
niño

****************************************
```

Negative base numbers

```
func EncodeNegBase(Num n, Num b { .~~ (-36 .. -2) }) {
    var out = []
    var r = 0
    while (n) {
        (n, r) = divmod(n, b)
        if (r < 0) {
            n += 1
            r -= b
        }
        out << r.base(-b)
    }
    return (out.join.flip || 0)
}

func DecodeNegBase(Str s, Num b { .~~ (-36 .. -2) }) {
    var total = 0
    for i,c in (s.flip.chars.pairs) {
        total += (Num(c, -b) * b**i)
    }
    return total
}

say (" 10 in base  -2: ", EncodeNegBase(10, -2))
say ("146 in base  -3: ", EncodeNegBase(146, -3))
say (" 15 in base -10: ", EncodeNegBase(15, -10))

say '-'*25

say ("11110 from base  -2: ", DecodeNegBase("11110", -2))
say ("21102 from base  -3: ", DecodeNegBase("21102", -3))
say ("  195 from base -10: ", DecodeNegBase("195",  -10))

say '-'*25

# Extra
say ("25334424 in base -31: ", EncodeNegBase(25334424, -31))
say ("sidef  from base -31: ", DecodeNegBase("sidef", -31))
```

Output:

```
 10 in base  -2: 11110
146 in base  -3: 21102
 15 in base -10: 195
-------------------------
11110 from base  -2: 10
21102 from base  -3: 146
  195 from base -10: 15
-------------------------
25334424 in base -31: sidef
sidef  from base -31: 25334424
```

Neighbour primes

```
500.primes.grep {|p| p * p.next_prime + 2 -> is_prime }.say
```

Output:

```
[3, 5, 7, 13, 19, 67, 149, 179, 229, 239, 241, 269, 277, 307, 313, 397, 401, 419, 439, 487]
```

Nested function

```
func make_list(separator = ') ') {

    var count = 1
    func make_item(item) {
        [count++, separator, item].join
    }

    <first second third>.map(make_item).join("\n")
}

say make_list('. ')
```

Output:

```
1. first
2. second
3. third
```

Next highest int from digits

```
func next_from_digits(n, b = 10) {

    var a = n.digits(b).flip

    while (a.next_permutation) {
        with (a.flip.digits2num(b)) { |t|
            return t if (t > n)
        }
    }

    return 0
}

say 'Next largest integer able to be made from these digits, or zero if no larger exists:'

for n in (
    0, 9, 12, 21, 12453, 738440, 3345333, 45072010,
    95322020, 982765431, 9589776899767587796600,
) {
    printf("%30s  ->  %s\n", n, next_from_digits(n))
}
```

Output:

```
Next largest integer able to be made from these digits, or zero if no larger exists:
                            0  ->  0
                            9  ->  0
                           12  ->  21
                           21  ->  0
                        12453  ->  12534
                       738440  ->  740348
                      3345333  ->  3353334
                     45072010  ->  45072100
                     95322020  ->  95322200
                    982765431  ->  983124567
        9589776899767587796600  ->  9589776899767587900667
```

Next special primes

```
func special_primes(upto) {

    var gap  = 0
    var prev = 2
    var list = [[prev, gap]]

    loop {
        var n = prev+gap
        n = n.next_prime
        break if (n > upto)
        gap = n-prev
        list << [n, gap]
        prev = n
    }

    return list
}

special_primes(1050).each_2d {|p,gap|
    say "#{'%4s' % p} #{'%4s' % gap}"
}
```

Output:

```
   2    0
   3    1
   5    2
  11    6
  19    8
  29   10
  41   12
  59   18
  79   20
 101   22
 127   26
 157   30
 191   34
 227   36
 269   42
 313   44
 359   46
 409   50
 461   52
 521   60
 587   66
 659   72
 733   74
 809   76
 887   78
 967   80
1049   82
```

Nice primes

```
func digital_root(n, base=10) {
    while (n.len(base) > 1) {
        n = n.sumdigits(base)
    }
    return n
}

say primes(500, 1000).grep { digital_root(_).is_prime }
```

Output:

```
[509, 547, 563, 569, 587, 599, 601, 617, 619, 641, 653, 659, 673, 677, 691, 709, 727, 743, 761, 797, 82
```

Non-continuous subsequences

```
func non_continuous(min, max, subseq=[], has_gap=false) {

    static current = []

    for i in (min .. max) {
        current.push(i)
        has_gap && subseq.append([current...])
        i < max && non_continuous(i.inc, max, subseq, has_gap)
        current.pop
        has_gap = current.len
    }

    return subseq
}

say non_continuous(1, 3)
say non_continuous(1, 4)
say non_continuous("a", "d")
```

Output:

```
[[1, 3]]
[[1, 2, 4], [1, 3], [1, 3, 4], [1, 4], [2, 4]]
[["a", "b", "d"], ["a", "c"], ["a", "c", "d"], ["a", "d"], ["b", "d"]]
```

Non-decimal radices/Convert

Built-in:

```
say 60272032366.base(36)    # convert number to string
say Number("rosetta", 36)   # convert string to number
```

User-defined:

```
static to = [@|'0'..'9', @|'a'..'z']
static from = Hash(to.pairs.map{@|_}.flip...)

func base_to(n, b) {
    var s = ""
    while (n) {
        s += to[n % b]
        n //= b
    }
    s.reverse
}

func base_from(n, b) {
    var t = 0
    n.each { |c| t = (b*t + from{c}) }
    t
}

say base_from("rosetta", 36)        # string to number
say base_to(60272032366, 36)        # number to string
```

Non-decimal radices/Input

```
var dec            = '0123459';
var hex_noprefix   = 'abcf123';
var hex_withprefix = '0xabcf123';
var oct_noprefix   = '7651';
var oct_withprefix = '07651';
var bin_noprefix   = '101011001';
var bin_withprefix = '0b101011001';

say dec.num;                    # => 123459
say hex_noprefix.hex;           # => 180154659
say hex_withprefix.hex;         # => 180154659
say oct_noprefix.oct;           # => 4009
say oct_withprefix.oct;         # => 4009
say bin_noprefix.bin;           # => 345
say bin_withprefix.bin;         # => 345
```

Non-decimal radices/Output

```
for i in (0 .. 33) {
    printf(" %6b %3o %2d %2X\n", ([i]*4)...)
}
```

Nth root

```
func nthroot(n, a, precision=1e-5) {
  var x = 1
  var prev = 0
  while (abs(prev-x) > precision) {
    prev = x
    x = float(((n-1)*prev + a/(prev**(n-1))) / n)
  }
  return x
}

say nthroot(5, 34)  # => 2.024397458501034082599817835297912829678314204
```

A minor optimization would be to calculate the successive *int(n-1)* square roots of a number, then raise the result to the power of *2**(int(n-1) / n)*.

```
func nthroot_fast(n, a, precision=1e-5) {
  { a = nthroot(2, a, precision) } * int(n-1)
  a ** (2**int(n-1) / n)
}

say nthroot_fast(5, 34, 1e-64)  # => 2.024397458499885042510817245541937419114621701 07
```

Null object

The absence of a value is represented by *nil*

```
var undefined          # initialized with an implicit nil
say undefined==nil     # true
say defined(nil)       # false
```

However, *nil* is not an object, so we can't call methods on it. Alternatively, Sidef provides the *null* object:

```
var null_obj = null         # initialize with a null value
say null_obj.is_a(null)     # true
say defined(null_obj)       # true
```

Number names

```
var l = frequire('Lingua::EN::Numbers')
say l.num2en(123456789)
```

Output:

```
one hundred and twenty-three million, four hundred and fifty-six thousand, seven hundred and eighty-nin
```

Number reversal game

```
var turn = 0
var jumble = @(1..9).bshuffle        # best-shuffle

for (turn; jumble != 1..9; ++turn) {
    printf("%2d: %s - Flip how many digits ? ", turn, jumble.join(' '))
    var d = read(Number) \\ break
    jumble[0 .. d-1] = [jumble[0 .. d-1]].reverse...
}

print "    #{jumble.join(' ')}\n"
print "You won in #{turn} turns.\n"
```

Numbers divisible by their individual digits, but not by the product of their digits.

```
^1000 -> grep {|n|
    n.digits.all {|d| d `divides` n } && !(n.digits.prod `divides` n)
}.say
```

Output:

```
[22, 33, 44, 48, 55, 66, 77, 88, 99, 122, 124, 126, 155, 162, 168, 184, 222, 244, 248, 264, 288, 324, 3
```

Numbers in base-16 representation that cannot be written with decimal digits

Simple solution:

```
1..500 -> grep { .digits(16).min >= 10 }.say
```

More efficient approach:

```
func generate_from_prefix(limit, p, base, digits) {

    var seq = [p]

    for d in (digits) {
        var t = [d, p...]
        if (t.digits2num(base) <= limit) {
            seq << __FUNC__(limit, t, base, digits)...
        }
    }

    return seq
}

func numbers_with_non_decimal_digits(limit, base = 10) {
    var digits = @(10..^base)
    digits.map  {|p| generate_from_prefix(limit, [p], base, digits)... }\
          .map  {|t| digits2num(t, base) }\
          .sort
}

say numbers_with_non_decimal_digits(500, 16)
```

Output:

```
[10, 11, 12, 13, 14, 15, 170, 171, 172, 173, 174, 175, 186, 187, 188, 189, 190, 191, 202, 203, 204, 205
```

Numbers in base 10 that are palindromic in bases 2, 4, and 16

```
say gather {
    for (var k = 0; k < 25_000; k = k.next_palindrome(16)) {
        take(k) if [2, 4].all{|b| k.is_palindrome(b) }
    }
}
```

Output:

```
[0, 1, 3, 5, 15, 17, 51, 85, 255, 257, 273, 771, 819, 1285, 1365, 3855, 4095, 4097, 4369, 12291, 13107,
```

Numbers which binary and ternary digit sum are prime

```
1..^200 -> grep {|n| [2,3].all { n.sumdigits(_).is_prime } }
```

Output:

```
[5, 6, 7, 10, 11, 12, 13, 17, 18, 19, 21, 25, 28, 31, 33, 35, 36, 37, 41, 47, 49, 55, 59, 61, 65, 67, 6
```

Numbers whose count of divisors is prime

```
var limit = 100_000
say "Positive integers under #{limit.commify} whose number of divisors is an odd prime:"

1..limit -> grep { !.is_prime && .sigma0.is_prime }.each_slice(10, {|*a|
    say a.map{'%6s' % _}.join(' ')
})
```

Output:

```
Positive integers under 100,000 whose number of divisors is an odd prime:
     4      9     16     25     49     64     81    121    169    289
   361    529    625    729    841    961   1024   1369   1681   1849
  2209   2401   2809   3481   3721   4096   4489   5041   5329   6241
  6889   7921   9409  10201  10609  11449  11881  12769  14641  15625
 16129  17161  18769  19321  22201  22801  24649  26569  27889  28561
 29929  32041  32761  36481  37249  38809  39601  44521  49729  51529
 52441  54289  57121  58081  59049  63001  65536  66049  69169  72361
 73441  76729  78961  80089  83521  85849  94249  96721  97969
```

Numbers with equal rises and falls

```
func isok(arr) {
    var diffs = arr.map_cons(2, {|a,b| a - b })
    diffs.count { .is_pos } == diffs.count { .is_neg }
}

var base = 10

with (200) {|n|
    say "First #{n} terms (base #{base}):"
    n.by { isok(.digits(base)) && .is_pos }.each_slice(20, {|*a|
        say a.map { '%3s' % _ }.join(' ')
    })
}

with (1e7) {|n|     # takes a very long time
    say "\nThe #{n.commify}-th term (base #{base}): #{
            n.th { isok(.digits(base)) && .is_pos }.commify}"
}
```

Output:

```
First 200 terms (base 10):
  1   2   3   4   5   6   7   8   9  11  22  33  44  55  66  77  88  99 101 102
103 104 105 106 107 108 109 111 120 121 130 131 132 140 141 142 143 150 151 152
153 154 160 161 162 163 164 165 170 171 172 173 174 175 176 180 181 182 183 184
185 186 187 190 191 192 193 194 195 196 197 198 201 202 203 204 205 206 207 208
209 212 213 214 215 216 217 218 219 222 230 231 232 240 241 242 243 250 251 252
253 254 260 261 262 263 264 265 270 271 272 273 274 275 276 280 281 282 283 284
285 286 287 290 291 292 293 294 295 296 297 298 301 302 303 304 305 306 307 308
309 312 313 314 315 316 317 318 319 323 324 325 326 327 328 329 333 340 341 342
343 350 351 352 353 354 360 361 362 363 364 365 370 371 372 373 374 375 376 380
381 382 383 384 385 386 387 390 391 392 393 394 395 396 397 398 401 402 403 404

The 10,000,000-th term (base 10): 41,909,002
```

Numbers with prime digits whose sum is 13

```
func generate_from_prefix(sum, p, base, digits) {

    var seq = [p]

    for d in (digits) {
        seq << __FUNC__(sum, [d, p...], base, digits)... if (p.sum+d <= sum)
    }

    return seq
}

func numbers_with_digitsum(sum, base = 10, digits = (base-1 -> primes)) {

    digits.map  {|p| generate_from_prefix(sum, [p], base, digits)... }\
          .map  {|t| digits2num(t, base) }\
          .grep {|t| t.sumdigits(base) == sum }\
          .sort
}

say numbers_with_digitsum(13)
```

Output:

```
[337, 355, 373, 535, 553, 733, 2227, 2272, 2335, 2353, 2533, 2722, 3235, 3253, 3325, 3352, 3523, 3532,
```

Numbers with same digit set in base 10 and base 16

Simple solution:

```
^1e5 -> grep { Set(.digits...) == Set(.digits(16)...) }.say
```

Recursively generate the numbers (2x faster):

```
func generate_from_prefix(limit, p, base, digits) {

    var seq = [p]

    for d in (digits) {
        var t = [d, p...]
        if (t.digits2num(base) <= limit) {
            seq << __FUNC__(limit, t, base, digits)...
        }
    }

    return seq
}

func numbers_with_same_digitset(limit, base = 10) {

    (1..9).map  {|p| generate_from_prefix(limit, [p], base, @(0..9))... }\
          .map  {|t| digits2num(t, base) }\
          .grep {|t| Set(t.digits...) == Set(t.digits(base)...) }\
          .sort\
          .prepend(0)
}

say numbers_with_same_digitset(1e5, 16)
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 53, 371, 913, 1040, 2080, 2339, 4100, 5141, 5412, 5441, 6182, 8200, 9241
```

Numeric separator syntax

Sidef allows underscores as a separator character in numeric inputs.

```
# Int
say 1_2_3;  # 123

# Binary Int
say 0b1_0_1_0_1; # 21

# Hexadecimal Int
say 0xa_bc_d; # 43981

# Rational
say 1_2_3_4.2_5; # 1234.25

# Rational in exponential notation
say 6.0_22e4; # 60220

# Apart from starting the number with an underscore, can be placed anywhere in the number.

say 1234_.25;       # 1234.25
say 1234._25;       # 1234.25
say 1234.25_;       # 1234.25
say 12__34.25;      # 1234.25
# say _1234.25;     # syntax error
```

Numerical integration

```
func sum(f, start, from, to) {
    var s = 0
    for i in RangeNum(start, to, from-start) {
        s += f(i)
    }
    return s
}

func leftrect(f, a, b, n) {
    var h = ((b - a) / n)
    h * sum(f, a, a+h, b-h)
}

func rightrect(f, a, b, n) {
    var h = ((b - a) / n)
    h * sum(f, a+h, a + 2*h, b)
}

func midrect(f, a, b, n) {
    var h = ((b - a) / n)
    h * sum(f, a + h/2, a + h + h/2, b - h/2)
}

func trapez(f, a, b, n) {
    var h = ((b - a) / n)
    h/2 * (f(a) + f(b) + sum({ f(_)*2 }, a+h, a + 2*h, b-h))
}

func simpsons(f, a, b, n) {
    var h = ((b - a) / n)
    var h2 = h/2

    var sum1 = f(a + h2)
    var sum2 = 0

    sum({|i| sum1 += f(i + h2); sum2 += f(i); 0 }, a+h, a+h+h, b-h)
    h/6 * (f(a) + f(b) + 4*sum1 + 2*sum2)
}

func tryem(label, f, a, b, n, exact) {
    say "\n#{label}\n   in [#{a}..#{b}] / #{n}"

    say('              exact result: ', exact)
    say('     rectangle method left: ', leftrect(f, a, b, n))
    say('    rectangle method right: ', rightrect(f, a, b, n))
    say('      rectangle method mid: ', midrect(f, a, b, n))
    say('composite trapezoidal rule: ', trapez(f, a, b, n))
    say('   quadratic simpsons rule: ', simpsons(f, a, b, n))
}

tryem('x^3', { _ ** 3 }, 0, 1, 100, 0.25)
tryem('1/x', { 1 / _ }, 1, 100, 1000, log(100))
tryem('x', { _ }, 0, 5_000, 5_000_000, 12_500_000)
tryem('x', { _ }, 0, 6_000, 6_000_000, 18_000_000)
```

Numerical integration/Adaptive Simpson's method

```
func adaptive_Simpson_quadrature(f, left, right, ε = 1e-9) {

    func quadrature_mid(l, lf, r, rf) {
        var mid = (l+r)/2
        var midf = f(mid)
        (mid, midf, abs(r-l)/6 * (lf + 4*midf + rf))
    }

    func recursive_asr(a, fa, b, fb, ε, whole, m, fm) {
        var (lm, flm, left)  = quadrature_mid(a, fa, m, fm)
        var (rm, frm, right) = quadrature_mid(m, fm, b, fb)
        var Δ = (left + right - whole)
        abs(Δ) <= 15*ε
            ? (left + right + Δ/15)
            : (__FUNC__(a, fa, m, fm, ε/2, left,  lm, flm) +
               __FUNC__(m, fm, b, fb, ε/2, right, rm, frm))
    }

    var (lf = f(left), rf = f(right))
    var (mid, midf, whole) = quadrature_mid(left, lf, right, rf)
    recursive_asr(left, lf, right, rf, ε, whole, mid, midf)
}

var (a = 0, b = 1)
var area = adaptive_Simpson_quadrature({ .sin }, a, b, 1e-15).round(-15)
say "Simpson's integration of sine from #{a} to #{b} ≈ #{area}"
```

Output:

```
Simpson's integration of sine from 0 to 1 ≈ 0.45969769413186
```

Numerical integration/Gauss-Legendre Quadrature

```
func legendre_pair((1), x) { (x, 1) }
func legendre_pair( n,  x) {
    var (m1, m2) = legendre_pair(n - 1, x)
    var u = (1 - 1/n)
    ((1 + u)*x*m1 - u*m2, m1)
}

func legendre((0), _) { 1 }
func legendre( n,  x) { [legendre_pair(n, x)][0] }

func legendre_prime({ .is_zero }, _) { 0 }
func legendre_prime({ .is_one  }, _) { 1 }

func legendre_prime(n, x) {
    var (m0, m1) = legendre_pair(n, x)
    (m1 - x*m0) * n / (1 - x**2)
}

func approximate_legendre_root(n, k) {
    # Approximation due to Francesco Tricomi
    var t = ((4*k - 1) / (4*n + 2))
    (1 - ((n - 1)/(8 * n**3))) * cos(Num.pi * t)
}

func newton_raphson(f, f_prime, r, eps = 2e-16) {
    loop {
        var dr = (-f(r) / f_prime(r))
        dr.abs >= eps || break
        r += dr
    }
    return r
}

func legendre_root(n, k) {
    newton_raphson(legendre.method(:call, n), legendre_prime.method(:call, n),
                   approximate_legendre_root(n, k))
}

func weight(n, r) { 2 / ((1 - r**2) * legendre_prime(n, r)**2) }

func nodes(n) {
    gather {
        take(Pair(0, weight(n, 0))) if n.is_odd
        { |i|
            var r = legendre_root(n, i)
            var w = weight(n, r)
            take(Pair(r, w), Pair(-r, w))
        }.each(1 .. (n >> 1))
    }
}

func quadrature(n, f, a, b, nds = nodes(n)) {
    func scale(x) { (x*(b - a) + a + b) / 2 }
    (b - a) / 2 * nds.sum { .second * f(scale(.first)) }
}

for i in (5..10, 20) {
    printf("Gauss-Legendre %2d-point quadrature ∫₋₃⁺³ exp(x) dx ≈ %.15f\n",
        i, quadrature(i, {.exp}, -3, +3))
}
```

Output:

```
Gauss-Legendre  5-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035577718385561
Gauss-Legendre  6-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035746975092341
Gauss-Legendre  7-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035749819726600
Gauss-Legendre  8-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035749854494515
Gauss-Legendre  9-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035749854817432
Gauss-Legendre 10-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035749854819787
Gauss-Legendre 20-point quadrature ∫₋₃⁺³ exp(x) dx ≈ 20.035749854819802
```

Odd and square numbers

```
var lo = 100
var hi = 1_000

say gather {
    for k in (lo.isqrt .. hi.isqrt) {
        take(k**2) if k.is_odd
    }
}
```

Output:

```
[121, 169, 225, 289, 361, 441, 529, 625, 729, 841, 961]
```

Odd squarefree semiprimes

```
func odd_squarefree_almost_primes(upto, k=2) {
    k.squarefree_almost_primes(upto).grep{.is_odd}
}

with (1e3) {|n|
    var list = odd_squarefree_almost_primes(n, 2)
    say "Found #{list.len} odd square-free semiprimes <= #{n.commify}:"
    say (list.first(10).join(', '), ', ..., ', list.last(10).join(', '))
}
```

Output:

```
Found 194 odd square-free semiprimes <= 1,000:
15, 21, 33, 35, 39, 51, 55, 57, 65, 69, ..., 951, 955, 959, 965, 973, 979, 985, 989, 993, 995
```

Odd word problem

Recursive solution:

```
func rev {
    (var c = STDIN.getc) \\ return()
    if (c ~~ /^[a-z]\z/i) {
        var r = rev()
        print c
        return r
    }
    return c
}

var (n=0, l=false)
while (defined(var c = STDIN.getc)) {
    var w = (c ~~ /^[a-z]\z/i)
    ++n if (w && !l)
    l = w
    if (n & 1) {
        print c
    } else {
        var r = rev()
        print(c, r)
        n = 0
        l = false
    }
}
```

Output:

```
$ echo 'what,is,the;meaning,of:life.' | sidef script.sf
what,si,the;gninaem,of:efil.

$ echo 'we,are;not,in,kansas;any,more.' | sidef script.sf
we,era;not,ni,kansas;yna,more.
```

Old lady swallowed a fly

```
var victims = [
    :fly:    "  I don't know why S—",
    :spider: "  That wriggled and jiggled and tickled inside her.",
    :bird:   "  How absurd, T!",
    :cat:    "  Fancy that, S!",
    :dog:    "  What a hog, T!",
    :goat:   "  She just opened her throat, and in walked the goat!",
    :cow:    "  I don't know how S!",
    :horse:  "  She's dead, of course...",
]

var history = ["I guess she'll die...\n"];

for victim,verse in victims {
    say "There was an old lady who swallowed a #{victim}...";

    verse.sub!(/\bS\b/, "she swallowed the #{victim}");
    verse.sub!(/\bT\b/, "to swallow a #{victim}!");

    say verse;
    verse ~~ /dead/ && break;

    history[0].sub!(/^X/, "She swallowed the #{victim}");
    history.each{.say};
    history.len < 5 && history.unshift(verse);
    history.unshift("X to catch the #{victim},");
}
```

Old Russian measure of length

```
func convert (magnitude, unit) {
     var factor = Hash(
        tochka     => 0.000254,
        liniya     => 0.00254,
        diuym      => 0.0254,
        vershok    => 0.04445,
        piad       => 0.1778,
        fut        => 0.3048,
        arshin     => 0.7112,
        sazhen     => 2.1336,
        versta     => 1066.8,
        milia      => 7467.6,
        centimeter => 0.01,
        meter      => 1,
        kilometer  => 1000,
    )

    var meters = (magnitude * factor{unit.lc})
    say("#{magnitude} #{unit} to:\n", '-' * 40)

    for u,f in (factor.sort_by { |_,v| v }) {
        printf("%10s: %s\n", u, meters / f) if (u != unit.lc)
    }
}

convert(1, 'meter')
say('')
convert(1, 'milia')
```

Output:

```
1 meter to:
---------------------------------------
    tochka: 3937.007874015748031496062992125984251968503937007874
    liniya: 393.70078740157480314960629921259842519685039370078 7
centimeter: 100
     diuym: 39.37007874015748031496062992125984251968503937007 9
   vershok: 22.49718785151856017997750281214848143982002249718 8
      piad: 5.624296962879640044994375703037120359955005624297
       fut: 3.280839895013123359580052493438320209973753280 84
    arshin: 1.406074240719910011248593925759280089988751406074
    sazhen: 0.468691413573303337082864641919760029996250468691
 kilometer: 0.001
    versta: 0.000937382827146606674165729283839520059992500937
     milia: 0.000133911832449515239166532754834217151427500134

1 milia to:
---------------------------------------
    tochka: 29400000
    liniya: 2940000
centimeter: 746760
     diuym: 294000
   vershok: 168000
      piad: 42000
       fut: 24500
    arshin: 10500
     meter: 7467.6
    sazhen: 3500
 kilometer: 7.4676
    versta: 7
```

One-dimensional cellular automata

```
var seq = "_###_##_#_#_#_#__#__"
var x = ''

loop {
    seq.tr!('01', '_#')
    say seq
    seq.tr!('_#', '01')
    seq.gsub!(/(?<=(.))(.)(?=(.))/, {|s1,s2,s3| s1 == s3 ? (s1 ? 1-s2.to_i : 0) : s2})
    (x != seq) && (x = seq) || break
}
```

Output:

```
_###_##_#_#_#_#__#__
_#_#####_#_#_#______
__##___##_#_#_______
__##___###_#________
__##___#_##_________
__##____###_________
__##____#_#_________
__##_____#__________
__##________________
```

```
class Automaton(rule, cells) {

    method init {
        rule = sprintf("%08b", rule).chars.map{.to_i}.flip
    }

    method next {
        var previous = cells.map{_}
        var len = previous.len
        cells[] = rule[
                previous.range.map { |i|
                    4*previous[i-1 % len] +
                    2*previous[i]         +
                      previous[i+1 % len]
                }...
            ]
    }

    method to_s {
        cells.map { _ ? '#' : ' ' }.join
    }
}

var size = 10
var auto = Automaton(
    rule: 104,
    cells: [(size/2).of(0)..., 111011010101.digits..., (size/2).of(0)...],
)

size.times {
    say "|#{auto}|"
    auto.next
}
```

Output:

```
|     ### ## # # #     |
|     # ##### # #      |
|      ##   ## #       |
|      ##   ###        |
|      ##   # #        |
|      ##    #         |
|      ##              |
|      ##              |
|      ##              |
|      ##              |
```

One of n lines in a file

```
func one_of_n(n) {
    var choice
    n.times { |i|
        choice = i if (1 > i.rand)
    }
    choice - 1
}

func one_of_n_test(n = 10, trials = 1_000_000) {
    var bins = []
    trials.times {
        bins[one_of_n(n)] := 0 ++
    }
    bins
}

say one_of_n_test()
```

Output:

```
99838 100843 99696 100078 99973 100350 100054 99495 99540 100133
```

Operator precedence

All operators in Sidef have the same precedence, which is controlled by lack of whitespace between the operands.

For example:

```
1+2 * 3+4    # means: (1+2) * (3+4)
```

See also the documentation on the precedence of operators.

Optional parameters

```
func table_sort(table, ordering: '<=>', column: 0, reverse: false) {
  if (reverse) {
    table.sort {|a,b| b[column].$ordering(a[column])}
  } else {
    table.sort {|a,b| a[column].$ordering(b[column])}
  }
}

# Quick example:
var table = [
  ["Ottowa", "Canada"],
  ["Washington", "USA"],
  ["Mexico City", "Mexico"],
]

say table_sort(table, column: 1)
```

Output:

```
[["Ottowa", "Canada"], ["Mexico City", "Mexico"], ["Washington", "USA"]]
```

Missing the point, we can also create and provide a custom method for sorting to *ordering*:

```
class String {
    method my_sort(arg) {
           (self.len <=> arg.len) ->
        || (self.lc <=> arg.lc)   ->
        || (self <=> arg)
    }
}

say table_sort(table, column: 1, ordering: 'my_sort')
```

Output:

```
[["Washington", "USA"], ["Ottowa", "Canada"], ["Mexico City", "Mexico"]]
```

Orbital elements

```
func orbital_state_vectors(
    semimajor_axis,
    eccentricity,
    inclination,
    longitude_of_ascending_node,
    argument_of_periapsis,
    true_anomaly
) {
    var (i, j, k) = (
        Vector(1, 0, 0),
        Vector(0, 1, 0),
        Vector(0, 0, 1),
    )

    func muladd(v1, x1, v2, x2) {
        (v1 * x1) + (v2 * x2)
    }

    func rotate(Ref i, Ref j, α) {
        (*i, *j) = (
            muladd(*i, +cos(α), *j, sin(α)),
            muladd(*i, -sin(α), *j, cos(α)),
        )
    }

    rotate(\i, \j, longitude_of_ascending_node)
    rotate(\j, \k, inclination)
    rotate(\i, \j, argument_of_periapsis)

    var l = (eccentricity == 1 ? 2*semimajor_axis
                               : semimajor_axis*(1 - eccentricity**2))

    var (c, s) = with(true_anomaly) { (.cos, .sin) }

    var r = l/(1 + eccentricity*c)
    var rprime = (s * r**2 / l)
    var position = muladd(i, c, j, s)*r

    var speed = muladd(i, rprime*c - r*s, j, rprime*s + r*c)
    speed /= speed.abs
    speed *= sqrt(2/r - 1/semimajor_axis)

    struct Result { position, speed }
    Result(position, speed)
}

for args in ([
    [1, 0.1, 0, 355/(113*6), 0, 0],
    [1, 0.1, Num.pi/18, Num.pi/6, Num.pi/4, 0]
]) {
    var r = orbital_state_vectors(args...)

    say "Arguments: #{args}:"
    say "Position : #{r.position}"
    say "Speed    : #{r.speed}\n"
}
```

Output:

```
Arguments: [1, 1/10, 0, 355/678, 0, 0]:
Position : [0.77942284339867983204217632822366303746470352798 6, 0.450000034653684237432302249506712706 8
Speed    : [-0.55277084096044375967327906231425954627708449409 7, 0.95742708317976153524620036861495209 5

Arguments: [1, 1/10, 0.17453292519943295769236907684886127134428718885 4, 0.52359877559829887307710723 05
Position : [0.23777128398220654779107184959165027147748809404, 0.86096026169771583466896627238269903921
Speed    : [-1.06193301748006004757467368094494935655538772696, 0.27585002056924950784645283033008548 93
```

Order disjoint list items

```
func dsort(m, n) {
    var h = Hash()
    n.each {|item| h{item} := 0 ++ }
    m.map  {|item| h{item} := 0 -- > 0 ? n.shift : item}
}

<<'EOT'.lines.each { |line|
        the cat sat on the mat  | mat cat
        the cat sat on the mat  | cat mat
        A B C A B C A B C       | C A C A
        A B C A B D A B E       | E A D A
        A B                     | B
        A B                     | B A
        A B B A                 | B A
EOT
        var (a, b) = line.split('|').map{.words}...
        say "#{a.join(' ')} | #{b.join(' ')} -> #{dsort(a.clone, b.clone).join(' ')}"
}
```

Output:

```
the cat sat on the mat | mat cat -> the mat sat on the cat
the cat sat on the mat | cat mat -> the cat sat on the mat
A B C A B C A B C | C A C A -> C B A C B A A B C
A B C A B D A B E | E A D A -> E B C A B D A B A
A B | B -> A B
A B | B A -> B A
A B B A | B A -> B A B A
```

Order two numerical lists

Built-in, via the comparison operator (`<=>`):

```
func ordered(a, b) {
    (a <=> b) < 0
}

for p in [
    Pair([1,2,4], [1,2,4]),
    Pair([1,2,4], [1,2]  ),
    Pair([1,2],   [1,2,4]),
] {
    var a = p.first
    var b = p.second
    var before = ordered(a, b)
    say "#{a} comes before #{b} : #{before}"
}
```

Output:

```
[1, 2, 4] comes before [1, 2, 4] : false
[1, 2, 4] comes before [1, 2] : false
[1, 2] comes before [1, 2, 4] : true
```

Ordered partitions

```
func part(_,    {.is_empty}) { [[]] }
func partitions({.is_empty}) { [[]] }

func part(s, args) {
  gather {
    s.combinations(args[0], { |*c|
      part(s - c, args.ft(1)).each{|r| take([c] + r) }
    })
  }
}

func partitions(args) {
  part(@(1..args.sum), args)
}

[[],[0,0,0],[1,1,1],[2,0,2]].each { |test_case|
  say "partitions #{test_case}:"
  partitions(test_case).each{|part| say part }
  print "\n"
}
```

Output:

```
partitions []:
[]

partitions [0, 0, 0]:
[[], [], []]

partitions [1, 1, 1]:
[[1], [2], [3]]
[[1], [3], [2]]
[[2], [1], [3]]
[[2], [3], [1]]
[[3], [1], [2]]
[[3], [2], [1]]

partitions [2, 0, 2]:
[[1, 2], [], [3, 4]]
[[1, 3], [], [2, 4]]
[[1, 4], [], [2, 3]]
[[2, 3], [], [1, 4]]
[[2, 4], [], [1, 3]]
[[3, 4], [], [1, 2]]
```

Ordered words

```
var words = [[]]
var file = %f'unixdict.txt'

file.open_r(\var fh, \var err) ->
    || die "Can't open file #{file}: $#{err}"

fh.each { |line|
    line.trim!
    if (line == line.sort) {
        words[line.length] := [] << line
    }
}

say words[-1].join(' ')
```

Output:

```
abbott accent accept access accost almost bellow billow biopsy chilly choosy choppy effort floppy gloss
```

Own digits power sum

```
func armstrong_numbers(n, base=10) {

    var D = @(^base)
    var P = D.map {|d| d**n }

    var list = []

    D.combinations_with_repetition(n, {|*c|
        var v = c.sum {|d| P[d] }
        if (v.digits(base).sort == c) {
            list.push(v)
        }
    })

    list.sort
}

for n in (3..10) {
    say ("For n = #{'%2d' % n}: ", armstrong_numbers(n))
}
```

Output:

```
For n =  3: [153, 370, 371, 407]
For n =  4: [1634, 8208, 9474]
For n =  5: [54748, 92727, 93084]
For n =  6: [548834]
For n =  7: [1741725, 4210818, 9800817, 9926315]
For n =  8: [24678050, 24678051, 88593477]
For n =  9: [146511208, 472335975, 534494836, 912985153]
For n = 10: [4679307774]
```

Padovan n-step number sequences

```
func padovan(N) {
    Enumerator({|callback|
        var n = 2
        var pn = [1, 1, 1]
        loop {
            pn << sum(pn[n-N .. (n++-1) -> grep { _ >= 0 }])
            callback(pn[-4])
        }
    })
}

for n in (2..8) {
    say "n = #{n} | #{padovan(n).first(25).join(' ')}"
}
```

Output:

```
n = 2 | 1 1 1 2 2 3 4 5 7 9 12 16 21 28 37 49 65 86 114 151 200 265 351 465 616
n = 3 | 1 1 1 2 3 4 6 9 13 19 28 41 60 88 129 189 277 406 595 872 1278 1873 2745 4023 5896
n = 4 | 1 1 1 2 3 5 7 11 17 26 40 61 94 144 221 339 520 798 1224 1878 2881 4420 6781 10403 15960
n = 5 | 1 1 1 2 3 5 8 12 19 30 47 74 116 182 286 449 705 1107 1738 2729 4285 6728 10564 16587 26044
n = 6 | 1 1 1 2 3 5 8 13 20 32 51 81 129 205 326 518 824 1310 2083 3312 5266 8373 13313 21168 33657
n = 7 | 1 1 1 2 3 5 8 13 21 33 53 85 136 218 349 559 895 1433 2295 3675 5885 9424 15091 24166 38698
n = 8 | 1 1 1 2 3 5 8 13 21 34 54 87 140 225 362 582 936 1505 2420 3891 6257 10061 16178 26014 41830
```

Palindrome dates

```
var palindates = Enumerator({ |f|
    var d = Date.strptime("2020-02-02", "%Y-%m-%d")
    loop {
        f(d) if d.strftime("%Y%m%d").is_palindrome
        d.add_days!(1)
    }
})

palindates.first(15).each { .strftime("%Y-%m-%d").say }
```

Output:

```
2020-02-02
2021-12-02
2030-03-02
2040-04-02
2050-05-02
2060-06-02
2070-07-02
2080-08-02
2090-09-02
2101-10-12
2110-01-12
2111-11-12
2120-02-12
2121-12-12
2130-03-12
```

Faster approach:

```
var palindates = gather {
    for y in (2020 .. 9999) {
        var (m, d) = Str(y).flip.last(4).split(2)...
        with ([y,m,d].join('-')) {|t|
            take(t) if Date.valid(t, "%Y-%m-%d")
        }
    }
}

say "Count of palindromic dates [2020..9999]: #{palindates.len}"

for a,b in ([
    ["First 15:", palindates.head(15)],
    ["Last 15:",  palindates.tail(15)]
]) {
    say ("\n#{a}\n", b.slices(5).map { .join("   ") }.join("\n"))
}
```

Output:

```
Count of palindromic dates [2020..9999]: 285

First 15:
2020-02-02   2021-12-02   2030-03-02   2040-04-02   2050-05-02
2060-06-02   2070-07-02   2080-08-02   2090-09-02   2101-10-12
2110-01-12   2111-11-12   2120-02-12   2121-12-12   2130-03-12

Last 15:
9170-07-19   9180-08-19   9190-09-19   9201-10-29   9210-01-29
9211-11-29   9220-02-29   9221-12-29   9230-03-29   9240-04-29
9250-05-29   9260-06-29   9270-07-29   9280-08-29   9290-09-29
```

Palindrome detection

Built-in

```
say "noon".is_palindrome    # true
```

Non-recursive

```
func palindrome(s) {
    s == s.reverse
}
```

Recursive

```
func palindrome(s) {
    if (s.len <= 1) {
        true
    }
    elsif (s.first != s.last) {
        false
    }
    else {
        __FUNC__(s.ft(1, -2))
    }
}
```

Palindrome pairs

```
func find_palindromes(arr, callback) {
    arr.len.variations(2, {|i,j|
        var w = (arr[i]+arr[j]).fc.gsub(/\W+/)
        callback(i, j) if (w == w.flip)
    })
}

var t = [
    "abcd","dcba","lls","s","sssll",
    "Mr. Owl ate","or a cat I saw?",
    " my metal worm","Was it a car ",
]

find_palindromes(t, {|i,j|
    say "(#{i}, #{j}) = (#{t[i].dump}, #{t[j].dump})"
})
```

Output:

```
(0, 1) = ("abcd", "dcba")
(1, 0) = ("dcba", "abcd")
(2, 4) = ("lls", "sssll")
(3, 2) = ("s", "lls")
(5, 7) = ("Mr. Owl ate", " my metal worm")
(8, 6) = ("Was it a car ", "or a cat I saw?")
```

Palindromic gapful numbers

Inspired from the C++ and Perl 6 entries.

```
class PalindromeGenerator (digit, base=10) {

    has power = base
    has after = (digit*power - 1)
    has even  = false

    method next {

        if (++after == power*(digit+1)) {
            power *= base if even
            after = digit*power
            even.not!
        }

        even ? (after*power*base + reverse(after, base))
             : (after*power + reverse(after/base, base))
    }
}

var task = [
    "(Required) First 20 gapful palindromes:",       { .first(20) }, 7,
    ,"\n(Required) 86th through 100th:",             { .first(1e2).last(15) }, 8,
    ,"\n(Optional) 991st through 1,000th:",          { .first(1e3).last(10) }, 10,
    ,"\n(Extra stretchy) 9,995th through 10,000th:", { .first(1e4).last(6) }, 12,
]

task.each_slice(3, {|title, f, w|
    say title
    for d in (1..9) {
        var k    = 11*d
        var iter = PalindromeGenerator(d)
        var arr  = f(^Inf->lazy.map { iter.next }.grep {|n| k `divides` n })
        say ("#{d}: ", arr.map{ "%*s" % (w, _) }.join(' '))
    }
})
```

Palindromic primes

```
func palindromic_primes(upto, base = 10) {
    var list = []
    for (var p = 2; p <= upto; p = p.next_palindrome(base)) {
        list << p if p.is_prime
    }
    return list
}

say palindromic_primes(1000)

for n in (1..10) {
    var count = palindromic_primes(10**n).len
    say "There are #{count} palindromic primes <= 10^#{n}"
}
```

Output:

```
[2, 3, 5, 7, 11, 101, 131, 151, 181, 191, 313, 353, 373, 383, 727, 757, 787, 797, 919, 929]
There are 4 palindromic primes <= 10^1
There are 5 palindromic primes <= 10^2
There are 20 palindromic primes <= 10^3
There are 20 palindromic primes <= 10^4
There are 113 palindromic primes <= 10^5
There are 113 palindromic primes <= 10^6
There are 781 palindromic primes <= 10^7
There are 781 palindromic primes <= 10^8
There are 5953 palindromic primes <= 10^9
There are 5953 palindromic primes <= 10^10
```

Palindromic primes in base 16

```
func palindromic_primes(upto, base = 10) {
    var list = []
    for (var p = 2; p <= upto; p = p.next_palindrome(base)) {
        list << p if p.is_prime
    }
    return list
}

var list = palindromic_primes(500, 16)

list.each {|p|
    say "#{'%3s' % p}_10 = #{'%3s' % p.base(16)}_16"
}
```

Output:

```
  2_10 =   2_16
  3_10 =   3_16
  5_10 =   5_16
  7_10 =   7_16
 11_10 =   b_16
 13_10 =   d_16
 17_10 =  11_16
257_10 = 101_16
337_10 = 151_16
353_10 = 161_16
401_10 = 191_16
433_10 = 1b1_16
449_10 = 1c1_16
```

Pandigital prime

```
func largest_pandigital_prime(base = 10, a = 1, b = base-1) {

    for n in (b `downto` 1) {

        var digits = @(a..n -> flip)

        if (base == 10) {   # check for divisibility by 3
            digits.sum % 3 == 0 && next
        }

        digits.permutations { |*p|
            var v = p.flip.digits2num(base)
            return v if v.is_prime
        }
    }

    return nil
}

say ("Max pandigital prime over [1, 9] is: ", largest_pandigital_prime(a: 1))
say ("Max pandigital prime over [0, 9] is: ", largest_pandigital_prime(a: 0))
```

Output:

```
Max pandigital prime over [1, 9] is: 7652413
Max pandigital prime over [0, 9] is: 76540231
```

Pangram checker

```
define Eng = 'a'..'z'
define Hex = 'a'..'f'
define Cyr = %w(а б в г д е ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я ё)

func pangram(str, alpha=Eng) {
    var lstr = str.lc
    alpha.all {|c| lstr.contains(c) }
}

say pangram("The quick brown fox jumps over the lazy dog.")
say pangram("My dog has fleas.")
say pangram("My dog has fleas.", Hex)
say pangram("My dog backs fleas.", Hex)
say pangram("Съешь же ещё этих мягких французских булок, да выпей чаю", Cyr)
```

Output:

```
true
false
false
true
true
```

Parallel brute force

```
func invert_sha256(hash) {

    var letters = @('a'..'z')

    var job = func (prefix, hash) {
        variations_with_repetition(letters, 4, {|*a|
            var s = join('', prefix, a...)
            return s if (s.sha256 == hash)
        })
        return nil
    }

    letters.map {|prefix|
        job.ffork(prefix, hash)
    }.each {|f|
        with (f.wait) { return _ }
    }
}

var tests = %w(
  1115dd800feaacefdf481f1f9070374a2a81e27880f187396db67958b207cbad
  3a7bd3e2360a3d29eea436fcfb7e44c735d117c42d1c1835420b6b9942dd4f1b
  74e1bb62f8dabb8125a58852b63bdf6eaef667cb56ac7f7cdba6d7305c50a22f
)

tests.each {|t|
    var phrase = invert_sha256(t)
    say "#{t} : #{phrase}"
}
```

Output:

```
1115dd800feaacefdf481f1f9070374a2a81e27880f187396db67958b207cbad : zyzzx
3a7bd3e2360a3d29eea436fcfb7e44c735d117c42d1c1835420b6b9942dd4f1b : apple
74e1bb62f8dabb8125a58852b63bdf6eaef667cb56ac7f7cdba6d7305c50a22f : mmmmm
```

Parallel calculations

The code uses the *prime_factors()* function defined in the "Prime decomposition" task.

```
var nums = [1275792312878611, 12345678915808973,
            1578070919762253, 14700694496703910,];

var factors = nums.map {|n| prime_factors.ffork(n) }.map { .wait }
say ((nums ~Z factors)->max_by {|m| m[1][0] })
```

Output:

```
$ time sidef parallel.sf
[1275792312878611, [11, 7369, 15739058129]]
sidef parallel.sf  24.46s user 0.02s system 158% cpu 15.436 total
```

Parsing/RPN calculator algorithm

```
var proggie = '3 4 2 * 1 5 - 2 3 ^ ^ / +'

class RPN(arr=[]) {

    method binop(op) {
        var x = arr.pop
        var y = arr.pop
        arr << y.(op)(x)
    }

    method run(p) {
        p.each_word { |w|
            say "#{w} (#{arr})"
            given (w) {
                when (/\d/) {
                    arr << Num(w)
                }
                when (<+ - * />) {
                    self.binop(w)
                }
                when ('^') {
                    self.binop('**')
                }
                default {
                    die "#{w} is bogus"
                }
            }
        }
        say arr[0]
    }
}

RPN.new.run(proggie)
```

Output:

```
3 ()
4 (3)
2 (3 4)
* (3 4 2)
1 (3 8)
5 (3 8 1)
- (3 8 1 5)
2 (3 8 -4)
3 (3 8 -4 2)
^ (3 8 -4 2 3)
^ (3 8 -4 8)
/ (3 8 65536)
+ (3 0.0001220703125)
3.0001220703125
```

Parsing/RPN to infix conversion

```
func p(pair, prec) {
    pair[0] < prec ? "( #{pair[1]} )" : pair[1]
}

func rpm_to_infix(string) {
    say "#{'='*17}\n#{string}"
    var stack = []
    string.each_word { |w|
        if (w ~~ /\d/) {
            stack << [9, Num(w)]
        }
        else {
            var y = stack.pop
            var x = stack.pop
            given(w) {
              when ('^')   { stack << [4, [p(x,5), w, p(y,4)].join(' ')] }
              when (<* />) { stack << [3, [p(x,3), w, p(y,3)].join(' ')] }
              when (<+ ->) { stack << [2, [p(x,2), w, p(y,2)].join(' ')] }
            }
            say stack
        }
    }
    say '-'*17
    stack.map{_[1]}
}

var tests = [
    '3 4 2 * 1 5 - 2 3 ^ ^ / +',
    '1 2 + 3 4 + ^ 5 6 + ^',
]

tests.each { say rpm_to_infix(_).join(' ') }
```

Output:

```
=================
3 4 2 * 1 5 - 2 3 ^ ^ / +
[[9, 3], [3, "4 * 2"]]
[[9, 3], [3, "4 * 2"], [2, "1 - 5"]]
[[9, 3], [3, "4 * 2"], [2, "1 - 5"], [4, "2 ^ 3"]]
[[9, 3], [3, "4 * 2"], [4, "( 1 - 5 ) ^ 2 ^ 3"]]
[[9, 3], [3, "4 * 2 / ( 1 - 5 ) ^ 2 ^ 3"]]
[[2, "3 + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3"]]
-----------------
3 + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3
=================
1 2 + 3 4 + ^ 5 6 + ^
[[2, "1 + 2"]]
[[2, "1 + 2"], [2, "3 + 4"]]
[[4, "( 1 + 2 ) ^ ( 3 + 4 )"]]
[[4, "( 1 + 2 ) ^ ( 3 + 4 )"], [2, "5 + 6"]]
[[4, "( ( 1 + 2 ) ^ ( 3 + 4 ) ) ^ ( 5 + 6 )"]]
-----------------
( ( 1 + 2 ) ^ ( 3 + 4 ) ) ^ ( 5 + 6 )
```

Parsing/Shunting-yard algorithm

```
var prec = Hash(
    '^' => 4,
    '*' => 3,
    '/' => 3,
    '+' => 2,
    '-' => 2,
    '(' => 1,
)

var assoc = Hash(
    '^' => 'right',
    '*' => 'left',
    '/' => 'left',
    '+' => 'left',
    '-' => 'left',
)

func shunting_yard(prog) {
    var inp = prog.words
    var ops = []
    var res = []

    func report (op) {
        printf("%25s    %-7s %10s %s\n",
            res.join(' '), ops.join(' '), op, inp.join(' '))
    }

    func shift  (t)  { report( "shift #{t}"); ops << t }
    func reduce (t)  { report("reduce #{t}"); res << t }

    while (inp) {
        given(var t = inp.shift) {
           when (/\d/) { reduce(t) }
           when ('(')  { shift(t) }
           when (')')  {
               while (ops) {
                 (var x = ops.pop) == '(' ? break : reduce(x)
               }
           }
           default {
                var newprec = prec{t}
                while (ops) {
                    var oldprec = prec{ops[-1]}

                    break if (newprec > oldprec)
                    break if ((newprec == oldprec) && (assoc{t} == 'right'))

                    reduce(ops.pop)
                }
                shift(t)
            }
        }
    }
    while (ops) { reduce(ops.pop) }
    return res
}

say shunting_yard('3 + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3').join(' ')
```

Output:

```
                                  reduce 3 + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3
                    3              shift + 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3
                    3    +        reduce 4 * 2 / ( 1 - 5 ) ^ 2 ^ 3
                  3 4    +         shift * 2 / ( 1 - 5 ) ^ 2 ^ 3
                  3 4    + *      reduce 2 / ( 1 - 5 ) ^ 2 ^ 3
                3 4 2    +        reduce * ( 1 - 5 ) ^ 2 ^ 3
              3 4 2 *    +         shift / ( 1 - 5 ) ^ 2 ^ 3
              3 4 2 *    + /       shift ( 1 - 5 ) ^ 2 ^ 3
              3 4 2 *    + / (    reduce 1 - 5 ) ^ 2 ^ 3
            3 4 2 * 1    + / (     shift - 5 ) ^ 2 ^ 3
            3 4 2 * 1    + / ( -  reduce 5 ) ^ 2 ^ 3
          3 4 2 * 1 5    + / (    reduce - ^ 2 ^ 3
        3 4 2 * 1 5 -    + /       shift ^ 2 ^ 3
        3 4 2 * 1 5 -    + / ^    reduce 2 ^ 3
      3 4 2 * 1 5 - 2    + / ^     shift ^ 3
      3 4 2 * 1 5 - 2    + / ^ ^  reduce 3
    3 4 2 * 1 5 - 2 3    + / ^    reduce ^
  3 4 2 * 1 5 - 2 3 ^    + /      reduce ^
3 4 2 * 1 5 - 2 3 ^ ^    +        reduce /
3 4 2 * 1 5 - 2 3 ^ ^ /            reduce +
3 4 2 * 1 5 - 2 3 ^ ^ / +
```

Partial function application

```
func fs(f) {
    func(*args) {
        args.map {f(_)}
    }
}

func double(n) { n  * 2 }
func square(n) { n ** 2 }

var fs_double = fs(double)
var fs_square = fs(square)

var s = @(0 .. 3)
say "fs_double(#{s}): #{fs_double(s...)}"
say "fs_square(#{s}): #{fs_square(s...)}"

s = [2, 4, 6, 8]
say "fs_double(#{s}): #{fs_double(s...)}"
say "fs_square(#{s}): #{fs_square(s...)}"
```

Output:

```
fs_double(0 1 2 3): 0 2 4 6
fs_square(0 1 2 3): 0 1 4 9
fs_double(2 4 6 8): 4 8 12 16
fs_square(2 4 6 8): 4 16 36 64
```

Partition an integer x into n primes

```
func prime_partition(num, parts) {

    if (parts == 1) {
        return (num.is_prime ? [num] : [])
    }

    num.primes.combinations(parts, {|*c|
        return c if (c.sum == num)
    })

    return []
}

var tests = [
    [   18, 2], [   19, 3], [   20,  4],
    [99807, 1], [99809, 1], [ 2017, 24],
    [22699, 1], [22699, 2], [22699,  3],
    [22699, 4], [40355, 3],
]

for num,parts (tests) {
    say ("Partition %5d into %2d prime piece" % (num, parts),
    parts == 1 ? ':  ' : 's: ', prime_partition(num, parts).join('+') || 'not possible')
}
```

Output:

```
Partition    18 into  2 prime pieces: 5+13
Partition    19 into  3 prime pieces: 3+5+11
Partition    20 into  4 prime pieces: not possible
Partition 99807 into  1 prime piece:  not possible
Partition 99809 into  1 prime piece:  99809
Partition  2017 into 24 prime pieces: 2+3+5+7+11+13+17+19+23+29+31+37+41+43+47+53+59+61+67+71+73+79+97+
Partition 22699 into  1 prime piece:  22699
Partition 22699 into  2 prime pieces: 2+22697
Partition 22699 into  3 prime pieces: 3+5+22691
Partition 22699 into  4 prime pieces: 2+3+43+22651
Partition 40355 into  3 prime pieces: 3+139+40213
```

Partition function P

Built-in:

```
say partitions(6666)   # very fast
```

User-defined:

```
func partitionsP(n) {
    func (n) is cached {

        n <= 1 && return n

        var a = sum(1..floor((sqrt(24*n + 1) + 1)/6), {|k|
            (-1)**(k-1) * __FUNC__(n - ((k*(3*k - 1)) >> 1))
        })

        var b = sum(1..ceil((sqrt(24*n + 1) - 7)/6), {|k|
            (-1)**(k-1) * __FUNC__(n - ((k*(3*k + 1)) >> 1))
        })

        a + b
    }(n+1)
}

var t = Time.micro

say partitionsP.map(0..25).join(' ')
say partitionsP(6666)

say ("Took %.4f seconds" % Time.micro-t)
```

Output:

```
1 1 2 3 5 7 11 15 22 30 42 56 77 101 135 176 231 297 385 490 627 792 1002 1255 1575 1958
193655306161707661080005073394486091998480950338405932486880600467114423441282418165863
Took 24.5225 seconds
```

Pascal's triangle

```
func pascal(rows) {
    var row = [1]
    { | n|
        say row.join(' ')
        row = [1, {|i| row[i] + row[i+1] }.map(0 .. n-2)..., 1]
    } << 1..rows
}

pascal(10)
```

Pascal's triangle/Puzzle

```
# set up triangle
var rows = 5
var tri = rows.of {|i| (i+1).of { Hash(x => 0, z => 0, v => 0, rhs => nil) } }
tri[0][0]{:rhs} = 151
tri[2][0]{:rhs} = 40
tri[4][0]{:x} = 1
tri[4][1]{:v} = 11
tri[4][2]{:x} = 1
tri[4][2]{:z} = 1
tri[4][3]{:v} = 4
tri[4][4]{:z} = 1

# aggregate from bottom to top
for row in (tri.len ^.. 1) {
    for col in (^tri[row-1]) {
        [:x, :z, :v].each { |key|
            tri[row-1][col]{key} = (tri[row][col]{key} + tri[row][col+1]{key})
        }
    }
}

# find equations
var eqn = gather {
    for r in tri {
        for c in r {
            take([c{:x}, c{:z}, c{:rhs} - c{:v}]) if defined(c{:rhs})
        }
    }
}

# print equations
say "Equations:"
say " x +  z = y"
for x,z,y in eqn { say "#{x}x + #{z}z = #{y}" }

# solve
var f = (eqn[0][1] / eqn[1][1])
{|i| eqn[0][i] -= (f * eqn[1][i]) } << ^3
f = (eqn[1][0] / eqn[0][0])
{|i| eqn[1][i] -= (f * eqn[0][i]) } << ^3

# print solution
say "Solution:"
var x = (eqn[0][2] / eqn[0][0])
var z = (eqn[1][2] / eqn[1][1])
var y = (x + z)
say "x=#{x}, y=#{y}, z=#{z}"
```

Output:

```
Equations:
 x +  z = y
7x + 7z = 91
2x + 1z = 18
Solution:
x=5, y=13, z=8
```

Pascal matrix generation

```
func grow_matrix(matrix, callback) {
    var m = matrix
    var s = m.len
    m[s][0] = callback(0, m[s-1][0], 0)
    m[0][s] = callback(m[0][s-1], 0, 0)
    {|i| m[i+1][s] = callback(m[i+1][s-1], m[i][s], m[i][s-1])} * (s-1)
    {|i| m[s][i+1] = callback(m[s][i], m[s-1][i+1], m[s-1][i])} * (s)
    return m
}

func transpose(matrix) {
    matrix[0].range.map{|i| matrix.map{_[i]} }
}

func madd_n_nw(m) { grow_matrix(m, ->(_, n, nw) { n + nw }) }
func madd_w_nw(m) { grow_matrix(m, ->(w, _, nw) { w + nw }) }
func madd_w_n(m)  { grow_matrix(m, ->(w, n, _)  { w + n  }) }

var functions = [madd_n_nw, madd_w_nw, madd_w_n].map { |f|
    func(n) {
        var r = [[1]]
        { f(r) } * n
        transpose(r)
    }
}

functions.map { |f|
    f(4).map { .map{ '%2s' % _ }.join(' ') }.join("\n")
}.join("\n\n").say
```

Output:

```
 1  1  1  1  1
 0  1  2  3  4
 0  0  1  3  6
 0  0  0  1  4
 0  0  0  0  1

 1  0  0  0  0
 1  1  0  0  0
 1  2  1  0  0
 1  3  3  1  0
 1  4  6  4  1

 1  1  1  1  1
 1  2  3  4  5
 1  3  6 10 15
 1  4 10 20 35
 1  5 15 35 70
```

Pathological floating point problems

Muller's sequence

```
func series (n) {
    var (u, v) = (2, -4)
    (n-2).times { (u, v) = (v, 111 - 1130/v + 3000/(v * u)) }
    return v
}

[(3..8)..., 20, 30, 50, 100].each {|n|
    printf("n = %3d -> %s\n", n, series(n))
}
```

Output:

```
n =   3 -> 18.5
n =   4 -> 9.37837837837837837837837837837837837837837837837837837784
n =   5 -> 7.80115273775216138328530259365994236311239193083 6
n =   6 -> 7.15441448097524935352789065386036202438123383819 7
n =   7 -> 6.80678473692363298394175659627200908762327670780 2
n =   8 -> 6.59263276870443839274200277636599482655298231773 5
n =  20 -> 6.04355211018926886777747736409754013318771500000 6
n =  30 -> 6.00678609303120575853055404795323970583307231443 8
n =  50 -> 6.00017584662718718894561402074719546952373517709 9
n = 100 -> 6.00000001931947792910408680340358571502435067543 7
```

The Chaotic Bank Society

```
var years = 25
var balance = (1 .. years+15 -> sum_by {|n| 1 / n! })
say "Starting balance, $(e-1): $#{balance}"
for i in (1..years) { balance = (i*balance - 1) }
printf("After year %d, you will have $%1.16g in your account.\n", years, balance)
```

Output:

```
Starting balance, $(e-1): $1.71828182845904523536028747135266249775724709 37
After year 25, you will have $0.03993872967323021 in your account.
```

Siegfried Rump's example

```
func f (a, b) {
    (333.75 * b**6) + (a**2 * ((11 * a**2 * b**2) -
      b**6 - (121 * b**4) - 2)) + (5.5 * b**8) + a/(2*b)
}

say f(77617.0, 33096.0)
```

Output:

```
-0.827396059946821368141165095479816291999033115784 4
```

Peano curve

Uses the LSystem class defined at Hilbert curve.

```
var rules = Hash(
    l => 'lFrFl-F-rFlFr+F+lFrFl',
    r => 'rFlFr+F+lFrFl-F-rFlFr',
)

var lsys = LSystem(
    width:  500,
    height: 500,

    xoff: -50,
    yoff: -50,

    len:   5,
    angle: 90,
    color: 'dark green',
)

lsys.execute('l', 4, "peano_curve.png", rules)
```

Output image: Peano curve

Pell's equation

```
func solve_pell(n) {

    var x = n.isqrt
    var y = x
    var z = 1
    var r = 2*x

    var (e1, e2) = (1, 0)
    var (f1, f2) = (0, 1)

    loop {

        y = (r*z - y)
        z = floor((n - y*y) / z)
        r = floor((x + y) / z)

        (e1, e2) = (e2, r*e2 + e1)
        (f1, f2) = (f2, r*f2 + f1)

        var A = (e2 + x*f2)
        var B = f2

        if (A**2 - n*B**2 == 1) {
            return (A, B)
        }
    }
}

for n in [61, 109, 181, 277] {
    var (x, y) = solve_pell(n)
    printf("x^2 - %3d*y^2 = 1 for x = %-21s and y = %s\n", n, x, y)
}
```

Output:

```
x^2 -  61*y^2 = 1 for x = 1766319049             and y = 226153980
x^2 - 109*y^2 = 1 for x = 158070671986249        and y = 15140424455100
x^2 - 181*y^2 = 1 for x = 2469645423824185801    and y = 183567298683461940
x^2 - 277*y^2 = 1 for x = 159150073798980475849 and y = 9562401173878027020
```

Penrose tiling

Using the LSystem class defined at Hilbert curve.

```
var rules = Hash(
    a => 'cE+dE----bE[-cE----aE]++',
    b => '+cE--dE[---aE--bE]+',
    c => '-aE++bE[+++cE++dE]-',
    d => '--cE++++aE[+dE++++bE]--bE',
    E => '',
)

var lsys = LSystem(
    width:  1000,
    height: 1000,

    scale: 1,
    xoff: -500,
    yoff: -500,

    len:   40,
    angle: 36,
    color: 'dark blue',
)

lsys.execute('[b]++[b]++[b]++[b]++[b]', 5, "penrose_tiling.png", rules)
```

Output image: Penrose tiling

Pentagram

Generates a SVG image to STDOUT.

```
func pentagram(dim=200, sides=5) {
    var pentagram = <<-EOT
    <?xml version="1.0" standalone="no" ?>
    <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd">
    <svg height="#{dim*2}" width="#{dim*2}" style="" xmlns="http://www.w3.org/2000/svg">
    <rect height="100%" width="100%" style="fill:black;" />
    EOT

    func pline(q) {
        <<-EOT
        <polyline points="#{{|n| '%0.3f' % n }.map(q, q[0], q[1]).join(' ')}"
        style="fill:blue; stroke:white; stroke-width:3;"
        transform="translate(#{dim}, #{dim}) rotate(-18)" />
        EOT
    }

    var v = {|k| 0.9 * dim * cis(k * Num.tau / sides) }.map(^sides)
    pentagram += pline([v[range(0, v.end, 2)], v[range(1, v.end, 2)]].map{.reals})
    pentagram += '</svg>'

    return pentagram
}

say pentagram()
```

Output:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd">
<svg height="400" width="400" style="" xmlns="http://www.w3.org/2000/svg">
<rect height="100%" width="100%" style="fill:black;" />
<polyline points="180.000 0.000 -145.623 105.801 55.623 -171.190 55.623 171.190 -145.623 -105.801
180.000 0.000"
style="fill:blue; stroke:white; stroke-width:3;"
transform="translate(200, 200) rotate(-18)" />
</svg>
```

Percentage difference between images

```
require('Imager')

func img_diff(a, b) {

    func from_file(name) {
         %O<Imager>.new(file => name)
    }

    func size(img) {
        (img.getwidth, img.getheight)
    }

    func pixel_diff(p1, p2) {
        [p1.rgba] »-« [p2.rgba] -> map { .abs }.sum
    }

    func read_pixel(img, x, y) {
        img.getpixel(x => x, y => y)
    }

    var(img1, img2) = (from_file(a), from_file(b))

    var(w1, h1) = size(img1)
    var(w2, h2) = size(img2)

    if ((w1 != w2) || (h1 != h2)) {
        return nil
    }

    var sum = 0
    for y=(^h1), x=(^w1) {
        sum += pixel_diff(read_pixel(img1, x, y), read_pixel(img2, x, y))
    }

    sum / (w1 * h1 * 255 * 3)
}

say 100*img_diff('Lenna50.jpg', 'Lenna100.jpg')
```

Output:

```
1.625593098160488153594771241830 07
```

Percolation/Mean run density

```
func R(n,p) {
    n.of { 1.rand < p ? 1 : 0}.sum
}

const t = 100
say ('t=', t)

Range(.1, .9, .2).each { |p|
    printf("p= %f, K(p)= %f\n", p, p*(1-p))
    [10, 100, 1000].each { |n|
        printf (" R(n, p)= %f\n", t.of { R(n, p) }.sum/n / t)
    }
}
```

Output:

```
t=100
p= 0.100000, K(p)= 0.090000
 R(n, p)= 0.099000
 R(n, p)= 0.105000
 R(n, p)= 0.099810
p= 0.300000, K(p)= 0.210000
 R(n, p)= 0.301000
 R(n, p)= 0.289800
 R(n, p)= 0.300720
p= 0.500000, K(p)= 0.250000
 R(n, p)= 0.481000
 R(n, p)= 0.501800
 R(n, p)= 0.498260
p= 0.700000, K(p)= 0.210000
 R(n, p)= 0.695000
 R(n, p)= 0.698400
 R(n, p)= 0.701220
p= 0.900000, K(p)= 0.090000
 R(n, p)= 0.910000
 R(n, p)= 0.898500
 R(n, p)= 0.899080
```

Percolation/Site percolation

```
class Percolate {

    has block = '▒'
    has water = '+'
    has pore  = ' '
    has grid  = 15
    has site  = []

    enum <DeadEnd, Up, Right, Down, Left>

    method direction(x, y) {
        ((site[y + 1][x] == pore) && Down ) ||
        ((site[y][x - 1] == pore) && Left ) ||
        ((site[y][x + 1] == pore) && Right) ||
        ((site[y - 1][x] == pore) && Up   ) ||
        DeadEnd
    }

    method move(dir, x, y) {
        given (dir) {
            when (Up)    { site[--y][x] = water }
            when (Down)  { site[++y][x] = water }
            when (Left)  { site[y][--x] = water }
            when (Right) { site[y][++x] = water }
        }
        return (x, y)
    }

    method percolate (prob  = 0.6) {
        site[0] = grid.of(pore)
        site[grid + 1] = grid.of(pore)

        for x = ^grid, y = 1..grid {
            site[y][x] = (1.rand < prob ? pore : block)
        }
```

```
        site[0][0] = water

        var stack = []
        var (x, y) = (0, 0)

        loop {
            if (var dir = self.direction(x, y)) {
                stack << [x, y]
                (x,y) = self.move(dir, x, y)
            }
            else {
                stack || return 0
                (x,y) = stack.pop...
            }
            return 1 if (y > grid)
        }
    }
}

var obj = Percolate()
say 'Sample percolation at 0.6'
obj.percolate(0.6)
obj.site.each { .join.say }
say ''

var tests = 100
say "Doing #{tests} trials at each porosity:"
for p in (0.1..1 `by` 0.1) {
    printf("p = %0.1f: %0.3f\n", p, tests.of { obj.percolate(p) }.sum / tests)
}
```

Output:

```
Sample percolation at 0.6
+
+    ▒▒▒  ▒ ▒▒
+  ▒  ▒   ▒ ▒
++++  ▒  ▒    ▒
▒+▒+++++    ▒ ▒
 ▒ ▒▒▒▒▒+
▒▒▒ ▒ ▒ +      ▒
▒     ▒+++ ▒  ▒
  ▒ ▒ ▒+▒+  ▒
▒ ▒    ▒ + ▒  ▒
 ▒   ▒ +++    ▒▒
▒▒▒▒ ▒▒▒▒+▒+▒ ▒ ▒
▒   ▒▒ + ▒    ▒
 ▒▒  ▒▒+++  ▒ ▒
     ▒ ▒▒+
▒   ▒ ▒  +
         +

Doing 100 trials at each porosity:
p = 0.1: 0.000
p = 0.2: 0.000
p = 0.3: 0.000
p = 0.4: 0.020
p = 0.5: 0.090
p = 0.6: 0.570
p = 0.7: 0.930
p = 0.8: 1.000
p = 0.9: 1.000
p = 1.0: 1.000
```

Perfect numbers

```
func is_perfect(n) {
    n.sigma == 2*n
}

for n in (1..10000) {
    say n if is_perfect(n)
}
```

Alternatively, a more efficient check for even perfect numbers:

```
func is_even_perfect(n) {

    var square = (8*n + 1)
    square.is_square || return false

    var t = ((square.isqrt + 1) / 2)
    t.is_smooth(2) || return false

    t-1 -> is_prime
}

for n in (1..10000) {
    say n if is_even_perfect(n)
}
```

Output:

```
6
28
496
8128
```

Perfect shuffle

```
func perfect_shuffle(deck) {
     deck/2 -> zip.flat
}

[8, 24, 52, 100, 1020, 1024, 10000].each { |size|
    var deck = @(1..size)
    var shuffled = deck

    var n = (1..Inf -> lazy.first {
        (shuffled = perfect_shuffle(shuffled)) == deck
    })

    printf("%5d cards: %4d\n", size, n)
}
```

Output:

```
    8 cards:    3
   24 cards:   11
   52 cards:    8
  100 cards:   30
 1020 cards: 1018
 1024 cards:   10
10000 cards:  300
```

Perfect totient numbers

```
func perfect_totient({.<=1}, sum=0) { sum }
func perfect_totient(     n, sum=0) { __FUNC__(var(t = n.euler_phi), sum + t) }

say (1..Inf -> lazy.grep {|n| perfect_totient(n) == n }.first(20))
```

Output:

```
[3, 9, 15, 27, 39, 81, 111, 183, 243, 255, 327, 363, 471, 729, 2187, 2199, 3063, 4359, 4375, 5571]
```

Perlin noise

```
const p = (%w'47 4G 3T 2J 2I F 3N D 5L 2N 2O 1H 5E 6H 7 69 3W 10 2V U 1X 3Y 8
2R 11 6O L A N 5A 6 44 6V 3C 6I 23 0 Q 5H 1Q 2M 70 63 5N 39 Z B W 1L 4X X 2G
6L 45 1K 2F 4U K 3H 3S 4R 4O 1W 4V 22 4L 1Z 3Q 3V 1C R 4M 25 42 4E 6F 2B 33 6D
3E 1O 5V 3P 6E 64 2X 2K 15 1J 1A 6T 14 6S 2U 3Z 1I 1T P 1R 4H 1 60 28 21 5T 24
3O 57 5S 2H I 4P 5K 5G 3R 3M 38 58 4F 2E 4K 2S 31 5I 4T 56 3 1S 1G 61 6A 6Y 3G
3F 5 5M 12 43 3A 3I 73 2A 2D 5W 5R 5Q 1N 6B 1B G 1M H 52 59 S 16 67 53 4Q 5X
3B 6W 48 2 18 4A 4J 1Y 65 49 2T 4B 4N 17 4S 9 3L M 13 71 J 2Q 30 32 27 35 68
6G 4Y 55 34 2W 62 6U 2P 6C 6Z Y 6Q 5D 6M 5U 40 C 5B 4Z 4I 6P 29 1F 41 6J 6X E
6N 2Z 1D 5C 5Y V 51 5J 2Y 4D 54 2C 5O 4W 37 3D 1E 19 3J 4 46 72 3U 6K 5P 2L 66
36 1V T O 20 6R 3X 3K 5F 26 1U 5Z 1P 4C 50' * 2 -> map {|n| Num(n, 36) })

func fade(n) { n * n * n * (n * (n * 6 - 15) + 10) }
func lerp(t, a, b) { a + t*(b-a) }

func grad(h, x, y, z) {
    h &= 15
    var u = (h < 8 ? x : y)
    var v = (h < 4 ? y : (h ~~ [12,14] ? x : z))
    (h&1 ? -u : u) + (h&2 ? -v : v)
}

func noise(x, y, z) {
    var(X, Y, Z) = [x, y, z].map { .floor & 255 }...
    var (u, v, w) = [x-=X, y-=Y, z-=Z].map { fade(_) }...
    var (AA, AB) = with(p[X]   + Y) {|i| (p[i] + Z, p[i+1] + Z) }
    var (BA, BB) = with(p[X+1] + Y) {|i| (p[i] + Z, p[i+1] + Z) }
    lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z  ),
                            grad(p[BA  ], x-1, y  , z  )),
                    lerp(u, grad(p[AB  ], x  , y-1, z  ),
                            grad(p[BB  ], x-1, y-1, z  ))),
            lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1),
                            grad(p[BA+1], x-1, y  , z-1)),
                    lerp(u, grad(p[AB+1], x  , y-1, z-1),
                            grad(p[BB+1], x-1, y-1, z-1))))
}

say noise(3.14, 42, 7)
```

Output:

```
0.136919958784
```

Permutation test

```
func statistic(ab, a) {
    var(sumab, suma) = (ab.sum, a.sum)
    suma/a.size - ((sumab-suma) / (ab.size-a.size))
}

func permutationTest(a, b) {
    var ab = (a + b)
    var tobs = statistic(ab, a)
    var under = (var count = 0)
    ab.combinations(a.len, {|*perm|
        statistic(ab, perm) <= tobs && (under += 1)
        count += 1
    })
    under * 100 / count
}

var treatmentGroup = [85, 88, 75, 66, 25, 29, 83, 39, 97]
var controlGroup   = [68, 41, 10, 49, 16, 65, 32, 92, 28, 98]
var under = permutationTest(treatmentGroup, controlGroup)
say ("under=%.2f%%, over=%.2f%%" % (under, 100 - under))
```

Output:

```
under=87.20%, over=12.80%
```

Permutations

Built-in

```
[1,2,3].permutations { |p|
    say p
}
```

Iterative

```
func forperm(callback, n) {
    var idx = @^n

    loop {
        callback(idx...)

        var p = n-1
        while (idx[p-1] > idx[p]) {--p}
        p == 0 && return()

        var d = p
        idx += idx.splice(p).reverse

        while (idx[p-1] > idx[d]) {++d}
        idx.swap(p-1, d)
    }

    return()
}

forperm({|*p| say p }, 3)
```

Recursive

```
func permutations(callback, set, perm=[]) {
    set || callback(perm)
    for i in ^set {
        __FUNC__(callback, [
            set[^i, i+1 ..^ set.len]
        ], [perm..., set[i]])
    }
    return()
}

permutations({|p| say p }, [0,1,2])
```

Output:

```
[0, 1, 2]
[0, 2, 1]
[1, 0, 2]
[1, 2, 0]
[2, 0, 1]
[2, 1, 0]
```

Permutations by swapping

```
func perms(n) {
   var perms = [[+1]]
   for x in (1..n) {
      var sign = -1
      perms = gather {
        for s,*p in perms {
          var r = (0 .. p.len)
          take((s < 0 ? r : r.flip).map {|i|
            [sign *= -1, p[^i], x, p[i..p.end]]
          }...)
        }
      }
   }
   perms
}

var n = 4
for p in perms(n) {
    var s = p.shift
    s > 0 && (s = '+1')
    say "#{p} => #{s}"
}
```

Output:

```
[1, 2, 3, 4] => +1
[1, 2, 4, 3] => -1
[1, 4, 2, 3] => +1
[4, 1, 2, 3] => -1
[4, 1, 3, 2] => +1
[1, 4, 3, 2] => -1
[1, 3, 4, 2] => +1
[1, 3, 2, 4] => -1
[3, 1, 2, 4] => +1
[3, 1, 4, 2] => -1
[3, 4, 1, 2] => +1
[4, 3, 1, 2] => -1
[4, 3, 2, 1] => +1
[3, 4, 2, 1] => -1
[3, 2, 4, 1] => +1
[3, 2, 1, 4] => -1
[2, 3, 1, 4] => +1
[2, 3, 4, 1] => -1
[2, 4, 3, 1] => +1
[4, 2, 3, 1] => -1
[4, 2, 1, 3] => +1
[2, 4, 1, 3] => -1
[2, 1, 4, 3] => +1
[2, 1, 3, 4] => -1
```

Permutations with repetitions

```
var k = %w(a b c)
var n = 2

cartesian([k] * n, {|*a| say a.join(' ') })
```

Output:

```
a a
a b
a c
b a
b b
b c
c a
c b
c c
```

Permutations with some identical elements

Simple implementation, by filtering out the duplicated permutations:

```
func permutations_with_some_identical_elements (reps) {
    reps.map_kv {|k,v| v.of(k+1)... }.permutations.uniq
}

say permutations_with_some_identical_elements([2,1]).map{.join}.join(' ')
say permutations_with_some_identical_elements([2,3,1]).map{.join}.join(' ')
```

Output:

```
112 121 211
112223 112232 112322 113222 121223 121232 121322 122123 122132 122213 122231 122312 122321 123122 12321
```

More efficient approach, by generating the permutations without duplicates:

```
func next_unique_perm (array) {

    var k = array.end
    return ([], false) if (k < 0)
    var i = k-1

    while ((i >= 0) && (array[i] >= array[i+1])) {
        --i
    }

    return (array.flip, false) if (i == -1)

    if (array[i+1] > array[k]) {
        array = [array.slice(0, i)..., array.slice(i+1, k).flip...]
    }

    var j = i+1
    while (array[i] >= array[j]) {
        j++
    }

    array.clone!
    array.swap(i,j)

    return (array, true)
}

func unique_permutations(array) {
    var perm  = array
    var perms = [perm]
    loop {
        (perm, var more) = next_unique_perm(perm)
        break if !more
        perms << perm
    }
    return perms
}

for arr in ([[1,1,2], [1,1,2,2,2,3], %w(A A B B B C)]) {
    say "\nPermutations with array = #{arr}:"
    say unique_permutations(arr).map{.join}.join(' ')
}
```

Output:

```
Permutations with array = [1, 1, 2]:
112 121 211

Permutations with array = [1, 1, 2, 2, 2, 3]:
112223 112232 112322 113222 121223 121232 121322 122123 122132 122213 122231 122312 122321 123122 12321

Permutations with array = ["A", "A", "B", "B", "B", "C"]:
AABBBC AABBCB AABCBB AACBBB ABABBC ABABCB ABACBB ABBABC ABBACB ABBBAC ABBBCA ABBCAB ABBCBA ABCABB ABCBA
```

Pernicious numbers

```
func is_pernicious(n) {
    n.sumdigits(2).is_prime
}

say is_pernicious.first(25).join(' ')
say is_pernicious.grep(888_888_877..888_888_888).join(' ')
```

Output:

```
3 5 6 7 9 10 11 12 13 14 17 18 19 20 21 22 24 25 26 28 31 33 34 35 36
888888877 888888878 888888880 888888883 888888885 888888886
```

Phrase reversals

```
var str    = "rosetta code phrase reversal";

say str.reverse;                            # reversed string
say str.words.map{.reverse}.join(' ');      # words reversed
say str.words.reverse.join(' ');            # word order reversed
```

Output:

```
lasrever esarhp edoc attesor
attesor edoc esarhp lasrever
reversal phrase code rosetta
```

Pi

```
func pi(callback) {
    var (q, r, t, k, n, l) = (1, 0, 1, 1, 3, 3)
    loop {
        if ((4*q + r - t) < n*t) {
            callback(n)
            static _dot = callback('.')
            var nr = 10*(r - n*t)
            n = ((10*(3*q + r)) // t - 10*n)
            q *= 10
            r = nr
        }
        else {
            var nr = ((2*q + r) * l)
            var nn = ((q*(7*k + 2) + r*l) // (t*l))
            q *= k
            t *= l
            l += 2
            k += 1
            n = nn
            r = nr
        }
    }
}

STDOUT.autoflush(true)
pi(func(digit){ print digit })
```

Pick random element

```
var arr = %w(north east south west)
say arr.rand
say arr.rand(2)
```

Output:

```
south
['west', 'south']
```

Pierpont primes

```
func smooth_generator(primes) {
    var s = primes.len.of { [1] }
    {
        var n = s.map { .first }.min
        { |i|
            s[i].shift if (s[i][0] == n)
            s[i] << (n * primes[i])
        } * primes.len
        n
    }
}

func pierpont_primes(n, k = 1) {
    var g = smooth_generator([2,3])
    1..Inf -> lazy.map { g.run + k }.grep { .is_prime }.first(n)
}

say "First 50 Pierpont primes of the 1st kind: "
say pierpont_primes(50, +1).join(' ')

say "\nFirst 50 Pierpont primes of the 2nd kind: "
say pierpont_primes(50, -1).join(' ')

for n in (250, 500, 1000) {
    var p = pierpont_primes(n, +1).last
    var q = pierpont_primes(n, -1).last
    say "\n#{n}th Pierpont prime of the 1st kind: #{p}"
    say "#{n}th Pierpont prime of the 2nd kind: #{q}"
}
```

Output:

```
First 50 Pierpont primes of the 1st kind:
2 3 5 7 13 17 19 37 73 97 109 163 193 257 433 487 577 769 1153 1297 1459 2593 2917 3457 3889 10369 1228

First 50 Pierpont primes of the 2nd kind:
2 3 5 7 11 17 23 31 47 53 71 107 127 191 383 431 647 863 971 1151 2591 4373 6143 6911 8191 8747 13121 1

250th Pierpont prime of the 1st kind: 625188645398570683335506940395 53
250th Pierpont prime of the 2nd kind: 411113117200095652589487508370 2271

500th Pierpont prime of the 1st kind: 222858821416333477371816280150118190656360950577385221282542 3873
500th Pierpont prime of the 2nd kind: 158245178672471201122056586003564712606492113901852574295199423 71

1000th Pierpont prime of the 1st kind: 692693147164396902504825580899971109615458182302320431071885374 2
1000th Pierpont prime of the 2nd kind: 130808875622796558124966904550677540789667321372943389238335302 7
```

Pig the dice game/Player

```
var (games=100) = ARGV.map{.to_i}...

define DIE = 1..6
define GOAL = 100

class Player(score=0, ante=0, rolls=0, strategy={false}) {
    method turn {
        rolls = 0
        ante  = 0
        loop {
            rolls++
            given (DIE.rand) { |roll|
                when (1) {
                    ante = 0
                    break
                }
                case (roll > 1) {
                    ante += roll
                }
            }
            ((score + ante >= GOAL) || strategy) && break
        }
        score += ante
    }
}

var players = []

# default, go-for-broke, always roll again
players[0] = Player()

# try to roll 5 times but no more per turn
players[1] = Player( strategy: { players[1].rolls >= 5 } )

# try to accumulate at least 20 points per turn
players[2] = Player( strategy: { players[2].ante > 20 } )

# random but 90% chance of rolling again
players[3] = Player( strategy: { 1.rand < 0.1 } )

# random but more conservative as approaches goal
players[4] = Player( strategy: { 1.rand < ((GOAL - players[4].score) * 0.6 / GOAL) } )

var wins = players.len.of(0)

games.times {
    var player = -1
    loop {
        player++
        var p = players[player % players.len]
        p.turn
        p.score >= GOAL && break
    }
    wins[player % players.len]++
    players.map{.score}.join("\t").say
    players.each { |p| p.score = 0 }
}

say "\nSCORES: for #{games} games"
say wins.join("\t")
```

Output:

```
0       0       67      101     19
0       88      100     9       14
0       81      66      100     24
0       56      103     8       27
0       102     70      17      27
0       79      101     29      36
0       100     71      56      31
0       62      104     28      34
103     19      24      6       5
0       49      101     24      19
0       60      22      101     0
0       20      101     30      34
...
...
...
0       101     69      26      91
0       87      101     30      54
0       84      100     17      64
0       52      24      102     17

SCORES: for 100 games
6       28      40      22      4
```

Piprimes

```
1..(prime(22)-1) -> map { .prime_count }.say
```

Output:

```
[0, 1, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 10, 10, 11, 11, 11
```

Pisano period

```
func pisano_period_pp(p,k) is cached {

    assert(k.is_pos,   "k = #{k} must be positive")
    assert(p.is_prime, "p = #{p} must be prime")

    var (a, b, n) = (0, 1, p**k)

    for k in (1..Inf) {
        (a, b) = (b, (a+b) % n)

        if ([a,b] == [0,1]) {
            return k
        }
    }
}

func pisano_period(n) {
    n.factor_map {|p,k| pisano_period_pp(p, k) }.lcm
}

say "Pisano periods for squares of primes p <= 15:"
say  15.primes.map {|p| pisano_period_pp(p, 2) }

say "\nPisano periods for primes p <= 180:"
say 180.primes.map {|p| pisano_period_pp(p, 1) }

say "\nPisano periods for integers n from 2 to 180:"
say pisano_period.map(2..180)
```

Output:

```
Pisano periods for squares of primes p <= 15:
[6, 24, 100, 112, 110, 364]

Pisano periods for primes p <= 180:
[3, 8, 20, 16, 10, 28, 36, 18, 48, 14, 30, 76, 40, 88, 32, 108, 58, 60, 136, 70, 148, 78, 168, 44, 196,

Pisano periods for integers n from 2 to 180:
[3, 8, 6, 20, 24, 16, 12, 24, 60, 10, 24, 28, 48, 40, 24, 36, 24, 18, 60, 16, 30, 48, 24, 100, 84, 72,
```

By assuming that **Wall-Sun-Sun primes** do not exist, we can compute the Pisano period more efficiently, as illustrated below on Fermat numbers **F_n = 2^(2^n) + 1**:

```
func pisano_period_pp(p, k=1) {
    (p - kronecker(5, p)).divisors.first_by {|d| fibmod(d, p) == 0 } * p**(k-1)
}

func pisano_period(n) {

    return 0 if (n <= 0)
    return 1 if (n == 1)

    var d = n.factor_map {|p,k| pisano_period_pp(p, k) }.lcm

    3.times {|k|
        var t = d<<k
        if ((fibmod(t, n) == 0) && (fibmod(t+1, n) == 1)) {
            return t
        }
    }
}

for k in (1..8) {
    say ("Pisano(F_#{k}) = ", pisano_period(2**(2**k) + 1))
}
```

Output:

```
Pisano(F_1) = 20
Pisano(F_2) = 36
Pisano(F_3) = 516
Pisano(F_4) = 14564
Pisano(F_5) = 2144133760
Pisano(F_6) = 4611702838532647040
Pisano(F_7) = 28356863910078205764000346543980814080
Pisano(F_8) = 385973630791054296284035667888885590056093947575123826968983748023917827891 2
```

Plasma effect

```
require('Imager')

class Plasma(width=400, height=400) {

    has img = nil

    method init {
        img = %O|Imager|.new(xsize => width, ysize => height)
    }

    method generate {
        for y=(^height), x=(^width) {
            var hue = (4 + sin(x/19) + sin(y/9) + sin((x+y)/25) + sin(hypot(x, y)/8))
            img.setpixel(x => x, y => y, color => Hash(hsv => [360 * hue / 8, 1, 1]))
        }
    }

    method save_as(filename) {
        img.write(file => filename)
    }
}

var plasma = Plasma(256, 256)
plasma.generate
plasma.save_as('plasma.png')
```

Output image

Playfair cipher

```
func playfair(key, from = 'J', to = (from == 'J' ? 'I' : '')) {

    func canon(str) {
        str.gsub(/[^[:alpha:]]/, '').uc.gsub(from, to)
    }

    var m = canon(key + ('A'..'Z' -> join)).chars.uniq.slices(5)

    var :ENC = gather {
        m.each { |r|
            for i,j in (^r ~X ^r) {
                i == j && next
                take(Pair("#{r[i]}#{r[j]}", "#{r[(i+1)%5]}#{r[(j+1)%5]}"))
            }
        }

        ^5 -> each { |k|
            var c = m.map {|a| a[k] }
            for i,j in (^c ~X ^c) {
                i == j && next
                take(Pair("#{c[i]}#{c[j]}", "#{c[(i+1)%5]}#{c[(j+1)%5]}"))
            }
        }

        cartesian([^5, ^5, ^5, ^5], {|i1,j1,i2,j2|
            i1 == i2 && next
            j1 == j2 && next
            take(Pair("#{m[i1][j1]}#{m[i2][j2]}", "#{m[i1][j2]}#{m[i2][j1]}"))
        })
    }.map { (.key, .value) }...

    var DEC = ENC.flip

    func enc(red) {
        gather {
            var str = canon(red)
            while (var m = (str =~ /(.)(?(?=\1)|(.?))/g)) {
                take("#{m[0]}#{m[1] == '' ? 'X' : m[1]}")
            }
        }.map { ENC{_} }.join(' ')
    }

    func dec(black) {
        canon(black).split(2).map { DEC{_} }.join(' ')
    }

    return(enc, dec)
}

var (encode, decode) = playfair('Playfair example')

var orig = "Hide the gold in...the TREESTUMP!!!"
say " orig:\t#{orig}"

var black = encode(orig)
say "black:\t#{black}"

var red = decode(black)
say "  red:\t#{red}"
```

Output:

```
 orig:  Hide the gold in...the TREESTUMP!!!
black:  BM OD ZB XD NA BE KU DM UI XM MO UV IF
  red:  HI DE TH EG OL DI NT HE TR EX ES TU MP
```

Playing cards

```
define Pip = <A 2 3 4 5 6 7 8 9 10 J Q K>;
define Suit = <♦ ♣ ♥ ♠>;

class Card(pip, suit) {
    method to_s { pip + suit }
}

class Deck(cards=[]) {

    method init {
        cards = gather {
            Pip.each { |p| Suit.each { |s| take(Card(p, s)) } }
        }
    }

    method shuffle {
        cards.shuffle!;
    }

    method deal { cards.shift };
    method to_s { cards.join(" ") };
}

var d = Deck();
say "Deck: #{d}";

var top = d.deal;
say "Top card: #{top}";

d.shuffle;
say "Deck, shuffled: #{d}";
```

Output:

```
Deck: A♦ A♣ A♥ A♠ 2♦ 2♣ 2♥ 2♠ 3♦ 3♣ 3♥ 3♠ 4♦ 4♣ 4♥ 4♠ 5♦ 5♣ 5♥ 5♠ 6♦ 6♣ 6♥ 6♠ 7♦ 7♣ 7♥ 7♠ 8♦ 8♣ 8♥ 8♠ 9
Top card: A♦
Deck, shuffled: 10♠ 2♠ 3♠ Q♥ 3♣ A♠ 6♠ 6♣ 9♣ 6♦ Q♦ 8♣ 4♦ 7♠ 10♦ 3♥ 4♠ 7♥ 8♠ 10♥ 10♣ 9♦ 5♠ Q♠ A♥ 4♥ J♥ Q♣
```

Plot coordinate pairs

```
require('GD::Graph::points')

var data = [
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
    [2.7, 2.8, 31.4, 38.1, 58.0, 76.2, 100.5, 130.0, 149.3, 180.0],
]

var graph = %s'GD::Graph::points'.new(400, 300)
var gd = graph.plot(data)

var format = 'png'
File("qsort-range.#{format}").write(gd.(format), :raw)
```

Pointers and references

A simple example of passing a variable-reference to a function:

```
func assign2ref(ref, value) {
    *ref = value
}

var x = 10
assign2ref(\x, 20)
say x      # x is now 20
```

Polymorphic copy

Object.dclone() makes a deep clone of any mutable object and returns it to the caller.

```
class T(value) {
    method display {
        say value
    }
}

class S(value) < T {
    method display {
        say value
    }
}

var obj1 = T("T")
var obj2 = S("S")
var obj3 = obj2.dclone          # make a deep clone of obj2

obj1.value = "foo"              # change the value of obj1
obj2.value = "bar"              # change the value of obj2

obj1.display                    # prints "foo"
obj2.display                    # prints "bar"
obj3.display                    # prints "S"
```

Polymorphism

```
class Point(x=0, y=0) {

}

class Circle(x=0, y=0, r=0) {

}

func pp(Point obj) {
    say "Point at #{obj.x},#{obj.y}"
}

func pp(Circle obj) {
    say "Circle at #{obj.x},#{obj.y} with radius #{obj.r}"
}
```

Example:

```
pp(Point.new)               # => Point at 0,0
var p = Point(1, 2)         # create a point
pp(p)                       # => Point at 1,2
say p.x                     # => 1
p.y += 1                    # add one to y
pp(p)                       # => Point at 1,3

var c = Circle(4,5,6)       # create a circle
var d = c.clone             # make a clone of it
d.r = 7.5                   # and change the radius to 7.5
pp(c)                       # => Circle at 4,5 with radius 6
pp(d)                       # => Circle at 4,5 with radius 7.5
```

Polynomial derivative

```
func derivative(f) {
    Poly(f.coeffs.map_2d{|e,k| [e-1, k*e] }.flat...)
}

var coeffs = [
    [5],
    [4,-3],
    [-1,6,5],
    [-4,3,-2,1],
    [-1, 6, 5],
    [1,1,0,-1,-1],
]

for c in (coeffs) {
    var poly = Poly(c.flip)
    var derv = derivative(poly)

    var d = { derv.coeff(_) }.map(0..derv.degree)

    say "Polynomial : #{'%20s' % c} = #{poly}"
    say "Derivative : #{'%20s' % d} = #{derv || 0}\n"
}
```

Output:

```
Polynomial :                  [5] = 5
Derivative :                  [0] = 0

Polynomial :              [4, -3] = -3*x + 4
Derivative :                 [-3] = -3

Polynomial :           [-1, 6, 5] = 5*x^2 + 6*x - 1
Derivative :              [6, 10] = 10*x + 6

Polynomial :       [-4, 3, -2, 1] = x^3 - 2*x^2 + 3*x - 4
Derivative :           [3, -4, 3] = 3*x^2 - 4*x + 3

Polynomial :           [-1, 6, 5] = 5*x^2 + 6*x - 1
Derivative :              [6, 10] = 10*x + 6

Polynomial :    [1, 1, 0, -1, -1] = -x^4 - x^3 + x + 1
Derivative :       [1, 0, -3, -4] = -4*x^3 - 3*x^2 + 1
```

Polynomial long division

```
func poly_long_div(rn, rd) {

    var n = rn.map{_}
    var gd = rd.len

    if (n.len >= gd) {
        return(gather {
            while (n.len >= gd) {
                var piv = n[0]/rd[0]
                take(piv)
                { |i|
                    n[i] -= (rd[i] * piv)
                } << ^(n.len `min` gd)
                n.shift
            }
        }, n)
    }

    return([0], rn)
}
```

Example:

```
func poly_print(c) {
    var l = c.len
    c.each_kv {|i, n|
        print n
        print("x^", (l - i - 1), " + ") if (i < l-1)
    }
    print "\n"
}

var poly = [
    Pair([1,-12,0,-42],  [1, -3]),
    Pair([1,-12,0,-42], [1,1,-3]),
    Pair(      [1,3,2],    [1,1]),
    Pair( [1,-4,6,5,3],  [1,2,1]),
]

poly.each { |pair|
    var (q, r) = poly_long_div(pair.first, pair.second)
    poly_print(q)
    poly_print(r)
    print "\n"
}
```

Output:

```
1x^2 + -9x^1 + -27
-123

1x^1 + -13
16x^1 + -81

1x^1 + 2
0

1x^2 + -6x^1 + 17
-23x^1 + -14
```

Polynomial regression

```
func regress(x, y, degree) {
    var A = Matrix.build(x.len, degree+1, {|i,j|
        x[i]**j
    })

    var B = Matrix.column_vector(y...)
    ((A.transpose * A)**(-1) * A.transpose * B).transpose[0]
}

func poly(x) {
    3*x**2 + 2*x + 1
}

var coeff = regress(
    10.of { _ },
    10.of { poly(_) },
    2
)

say coeff
```

Output:

```
[1, 2, 3]
```

Polynomial synthetic division

```
func extended_synthetic_division(dividend, divisor) {
    var end = divisor.end
    var out = dividend.clone
    var normalizer = divisor[0]

    for i in ^(dividend.len - end) {
        out[i] /= normalizer
        var coef = out[i]
        if (coef != 0) {
            for j in (1 .. end) {
                out[i+j] += -(divisor[j] * coef)
            }
        }
    }

    var remainder = out.splice(-end)
    var quotient = out

    return(quotient, remainder)
}

var (n, d) = ([1, -12, 0, -42], [1, -3])
print("  %s / %s =" % (n, d))
print(" %s remainder %s\n" % extended_synthetic_division(n, d))
```

Output:

```
  [1, -12, 0, -42] / [1, -3] = [1, -9, -27] remainder [-123]
```

Population count

Built-in:

```
say popcount(42)
```

User-defined:

```
func population_count(n) { n.as_bin.count('1') }
say "#{0..29 «**« 3 «call« population_count -> join(' ')}"

var numbers = 60.of { |i|
    [i, population_count(i)]
}

say "Evil:   #{numbers.grep{_[1] %% 2}.map{.first}.join(' ')}"
say "Odious: #{numbers.grep{_[1] &  1}.map{.first}.join(' ')}"
```

Output:

```
1 2 2 4 3 6 6 5 6 8 9 13 10 11 14 15 11 14 14 17 17 20 19 22 16 18 24 30 25 25
Evil:   0 3 5 6 9 10 12 15 17 18 20 23 24 27 29 30 33 34 36 39 40 43 45 46 48 51 53 54 57 58
Odious: 1 2 4 7 8 11 13 14 16 19 21 22 25 26 28 31 32 35 37 38 41 42 44 47 49 50 52 55 56 59
```

Positive decimal integers with the digit 1 occurring exactly twice

```
say (1..1000 -> grep { .digits.count { _ == 1 } == 2 })
```

Output:

```
[11, 101, 110, 112, 113, 114, 115, 116, 117, 118, 119, 121, 131, 141, 151, 161, 171, 181, 191, 211, 311
```

Power set

```
var arr = %w(a b c)
for i in (0 .. arr.len) {
    say arr.combinations(i)
}
```

Output:

```
[[]]
[["a"], ["b"], ["c"]]
[["a", "b"], ["a", "c"], ["b", "c"]]
[["a", "b", "c"]]
```

Powerful numbers

Generation

```
func powerful(n, k=2) {

    var list = []

    func (m,r) {
        if (r < k) {
            list << m
            return nil
        }
        for a in (1 .. iroot(idiv(n,m), r)) {
            if (r > k) {
                a.is_coprime(m) || next
                a.is_squarefree || next
            }
            __FUNC__(m * a**r, r-1)
        }
    }(1, 2*k - 1)

    return list.sort
}

for k in (2..10) {
    var a = powerful(10**k, k)
    var h = a.head(5).join(', ')
    var t = a.tail(5).join(', ')
    printf("For k=%-2d there are %d k-powerful numbers <= 10^k: [%s, ..., %s]\n", k, a.len, h, t)
}
```

Output:

```
For k=2  there are 14 k-powerful numbers <= 10^k: [1, 4, 8, 9, 16, ..., 49, 64, 72, 81, 100]
For k=3  there are 20 k-powerful numbers <= 10^k: [1, 8, 16, 27, 32, ..., 625, 648, 729, 864, 1000]
For k=4  there are 25 k-powerful numbers <= 10^k: [1, 16, 32, 64, 81, ..., 5184, 6561, 7776, 8192, 1000
For k=5  there are 32 k-powerful numbers <= 10^k: [1, 32, 64, 128, 243, ..., 65536, 69984, 78125, 93312
For k=6  there are 38 k-powerful numbers <= 10^k: [1, 64, 128, 256, 512, ..., 559872, 746496, 823543, 8
For k=7  there are 46 k-powerful numbers <= 10^k: [1, 128, 256, 512, 1024, ..., 7558272, 8388608, 89579
For k=8  there are 52 k-powerful numbers <= 10^k: [1, 256, 512, 1024, 2048, ..., 60466176, 67108864, 80
For k=9  there are 59 k-powerful numbers <= 10^k: [1, 512, 1024, 2048, 4096, ..., 644972544, 725594112,
For k=10 there are 68 k-powerful numbers <= 10^k: [1, 1024, 2048, 4096, 8192, ..., 7739670528, 85899345
```

Counting

```
func powerful_count(n, k=2) {

    var count = 0

    func (m,r) {
        if (r <= k) {
            count += iroot(idiv(n,m), r)
            return nil
        }
        for a in (1 .. iroot(idiv(n,m), r)) {
            a.is_coprime(m) || next
            a.is_squarefree || next
            __FUNC__(m * a**r, r-1)
        }
    }(1, 2*k - 1)

    return count
}

for k in (2..10) {
    var a = (k+10).of {|j| powerful_count(10**j, k) }
    printf("Number of %2d-powerful numbers <= 10^j, for 0 <= j < #{k+10}: %s\n", k, a)
}
```

Output:

```
Number of  2-powerful numbers <= 10^j, for 0 <= j < 12: [1, 4, 14, 54, 185, 619, 2027, 6553, 21044, 672
Number of  3-powerful numbers <= 10^j, for 0 <= j < 13: [1, 2, 7, 20, 51, 129, 307, 713, 1645, 3721, 83
Number of  4-powerful numbers <= 10^j, for 0 <= j < 14: [1, 1, 5, 11, 25, 57, 117, 235, 464, 906, 1741,
Number of  5-powerful numbers <= 10^j, for 0 <= j < 15: [1, 1, 3, 8, 16, 32, 63, 117, 211, 375, 659, 11
Number of  6-powerful numbers <= 10^j, for 0 <= j < 16: [1, 1, 2, 6, 12, 21, 38, 70, 121, 206, 335, 551
Number of  7-powerful numbers <= 10^j, for 0 <= j < 17: [1, 1, 1, 4, 10, 16, 26, 46, 77, 129, 204, 318,
Number of  8-powerful numbers <= 10^j, for 0 <= j < 18: [1, 1, 1, 3, 8, 13, 19, 32, 52, 85, 135, 211, 3
Number of  9-powerful numbers <= 10^j, for 0 <= j < 19: [1, 1, 1, 2, 6, 11, 16, 24, 38, 59, 94, 145, 21
Number of 10-powerful numbers <= 10^j, for 0 <= j < 20: [1, 1, 1, 1, 5, 9, 14, 21, 28, 43, 68, 104, 155
```

Practical numbers

Built-in:

```
say is_practical(2**128 + 1)   #=> false
say is_practical(2**128 + 4)   #=> true
```

Slow implementation (as the task requires):

```
func is_practical(n) {

    var set = Set()

    n.divisors.grep { _ < n }.subsets {|*a|
        set << a.sum
    }

    1..n-1 -> all { set.has(_) }
}

var from = 1
var upto = 333

var list = (from..upto).grep { is_practical(_) }

say "There are #{list.len} practical numbers in the range #{from}..#{upto}."
say "#{list.first(10).join(', ')} ... #{list.last(10).join(', ')}\n"

for n in ([666, 6666, 66666]) {
    say "#{'%5s' % n } is practical? #{is_practical(n)}"
}
```

Efficient algorithm:

```
func is_practical(n) {

    n.is_odd && return (n == 1)
    n.is_pos || return false

    var p = 1
    var f = n.factor_exp

    f.each_cons(2, {|a,b|
        p *= sigma(a.head**a.tail)
        b.head > (1 + p) && return false
    })

    return true
}
```

Output:

```
There are 77 practical numbers in the range 1..333.
1, 2, 4, 6, 8, 12, 16, 18, 20, 24 ... 288, 294, 300, 304, 306, 308, 312, 320, 324, 330

  666 is practical? true
 6666 is practical? true
66666 is practical? false
```

Price fraction

```
var table = <<'EOT'.lines.map { .words.grep{.is_numeric}.map{.to_n} }
>=  0.00  <  0.06  :=  0.10
>=  0.06  <  0.11  :=  0.18
>=  0.11  <  0.16  :=  0.26
>=  0.16  <  0.21  :=  0.32
>=  0.21  <  0.26  :=  0.38
>=  0.26  <  0.31  :=  0.44
>=  0.31  <  0.36  :=  0.50
>=  0.36  <  0.41  :=  0.54
>=  0.41  <  0.46  :=  0.58
>=  0.46  <  0.51  :=  0.62
>=  0.51  <  0.56  :=  0.66
>=  0.56  <  0.61  :=  0.70
>=  0.61  <  0.66  :=  0.74
>=  0.66  <  0.71  :=  0.78
>=  0.71  <  0.76  :=  0.82
>=  0.76  <  0.81  :=  0.86
>=  0.81  <  0.86  :=  0.90
>=  0.86  <  0.91  :=  0.94
>=  0.91  <  0.96  :=  0.98
>=  0.96  <  1.01  :=  1.00
EOT

func price(money) {
    table.each { |row|
           (row[0] <= money) ->
        && (row[1] >  money) ->
        && return row[2]
    }
    die "Out of range"
}

for n in %n(0.3793 0.4425 0.0746 0.6918 0.2993 0.5486 0.7848 0.9383 0.2292) {
    say price(n)
}
```

Output:

```
0.54
0.58
0.18
0.78
0.44
0.66
0.86
0.98
0.38
```

Primality by trial division

```
func is_prime(a) {
  given (a) {
    when (2)                  { true  }
    case (a <= 1 || a.is_even) { false }
    default                   { 3 .. a.isqrt -> any { .divides(a) } -> not }
  }
}
```

Primality by Wilson's theorem

```
func is_wilson_prime_slow(n) {
    n > 1 || return false
    (n-1)! % n == n-1
}

func is_wilson_prime_fast(n) {
    n > 1 || return false
    factorialmod(n-1, n) == n-1
}

say 25.by(is_wilson_prime_slow)     #=> [2, 3, 5, ..., 83, 89, 97]
say 25.by(is_wilson_prime_fast)     #=> [2, 3, 5, ..., 83, 89, 97]

say is_wilson_prime_fast(2**43 - 1)   #=> false
say is_wilson_prime_fast(2**61 - 1)   #=> true
```

Prime conspiracy

```
var primes = (^Inf -> lazy.grep{.is_prime})

var upto = 1e6
var conspiracy = Hash()

primes.first(upto+1).reduce { |a,b|
    var d = b%10
    conspiracy{"#{a} → #{d}"} := 0 ++
    d
}

for k,v in (conspiracy.sort_by{|k,_v| k }) {
    printf("%s count: %6s\tfrequency: %2.2f %\n", k, v.commify, v / upto * 100)
}
```

Output:

```
1 → 1 count: 42,853     frequency: 4.29 %
1 → 3 count: 77,475     frequency: 7.75 %
1 → 7 count: 79,453     frequency: 7.95 %
1 → 9 count: 50,153     frequency: 5.02 %
2 → 3 count:      1     frequency: 0.00 %
3 → 1 count: 58,255     frequency: 5.83 %
3 → 3 count: 39,668     frequency: 3.97 %
3 → 5 count:      1     frequency: 0.00 %
3 → 7 count: 72,828     frequency: 7.28 %
3 → 9 count: 79,358     frequency: 7.94 %
5 → 7 count:      1     frequency: 0.00 %
7 → 1 count: 64,230     frequency: 6.42 %
7 → 3 count: 68,595     frequency: 6.86 %
7 → 7 count: 39,603     frequency: 3.96 %
7 → 9 count: 77,586     frequency: 7.76 %
9 → 1 count: 84,596     frequency: 8.46 %
9 → 3 count: 64,371     frequency: 6.44 %
9 → 7 count: 58,130     frequency: 5.81 %
9 → 9 count: 42,843     frequency: 4.28 %
```

Prime decomposition

Built-in:

```
say factor(536870911)      #=> [233, 1103, 2089]
say factor_exp(536870911)  #=> [[233, 1], [1103, 1], [2089, 1]]
```

Trial division:

```
func prime_factors(n) {
    return [] if (n < 1)
    gather {
        while (!(n & 1)) {
            n >>= 1
            take(2)
        }
        var p = 3
        while ((n > 1) && (p*p <= n)) {
            while (n %% p) {
                n //= p
                take(p)
            }
            p += 2
        }
        take(n) if (n > 1)
    }
}
```

Calling the function:

```
say prime_factors(536870911)
```

Output:

```
[233, 1103, 2089]
```

Prime numbers p for which the sum of primes less than or equal to p is prime

```
func primes_with_prime_sum(n, callback) {
    var s = 0
    n.each_prime {|p|
        s += p
        callback(p, s) if s.is_prime
    }
}

primes_with_prime_sum(1000, {|p,s|
    say "prime: #{'%3s' % p}   prime sum: #{'%5s' % s}"
})
```

Output:

```
prime:   2   prime sum:     2
prime:   3   prime sum:     5
prime:   7   prime sum:    17
prime:  13   prime sum:    41
prime:  37   prime sum:   197
prime:  43   prime sum:   281
prime: 281   prime sum:  7699
prime: 311   prime sum:  8893
prime: 503   prime sum: 22039
prime: 541   prime sum: 24133
prime: 557   prime sum: 25237
prime: 593   prime sum: 28697
prime: 619   prime sum: 32353
prime: 673   prime sum: 37561
prime: 683   prime sum: 38921
prime: 733   prime sum: 43201
prime: 743   prime sum: 44683
prime: 839   prime sum: 55837
prime: 881   prime sum: 61027
prime: 929   prime sum: 66463
prime: 953   prime sum: 70241
```

Prime numbers p which sum of prime numbers less or equal to p is prime

```
func primes_with_prime_sum(n, callback) {
    var s = 0
    n.each_prime {|p|
        s += p
        callback(p, s) if s.is_prime
    }
}

primes_with_prime_sum(1000, {|p,s|
    say "prime: #{'%3s' % p}   prime sum: #{'%5s' % s}"
})
```

Output:

```
prime:   2   prime sum:     2
prime:   3   prime sum:     5
prime:   7   prime sum:    17
prime:  13   prime sum:    41
prime:  37   prime sum:   197
prime:  43   prime sum:   281
prime: 281   prime sum:  7699
prime: 311   prime sum:  8893
prime: 503   prime sum: 22039
prime: 541   prime sum: 24133
prime: 557   prime sum: 25237
prime: 593   prime sum: 28697
prime: 619   prime sum: 32353
prime: 673   prime sum: 37561
prime: 683   prime sum: 38921
prime: 733   prime sum: 43201
prime: 743   prime sum: 44683
prime: 839   prime sum: 55837
prime: 881   prime sum: 61027
prime: 929   prime sum: 66463
prime: 953   prime sum: 70241
```

Prime numbers which contain 123

```
func numbers_with_subdigits(upto, base = 10, s = 123.digits(base)) {
    Enumerator({|callback|
        for k in (0 .. base**(upto.len(base) - s.len)) {

            var d = k.digits(base)

            for i in (0 .. d.len) {
                var n = d.clone.insert(i, s...).digits2num(base)
                callback(n) if (n <= upto)
            }

            var z = d.clone.insert(d.len, s...)
            loop {
                var n = z.insert(d.len, 0).digits2num(base)
                (n <= upto) ? callback(n) : break
            }
        }
    })
}

say "Decimal primes under 100,000 which contain '123':"
numbers_with_subdigits(1e5).grep { .is_prime }.sort.each_slice(10, {|*a|
    say a.map { '%6s' % _ }.join(' ')
})

say ''

for n in (4..8) {
    var count = numbers_with_subdigits(10**n).grep { .is_prime }.len
    say "Found #{'%6s' % count.commify} such primes < 10^#{n}"
}
```

Output:

```
Decimal primes under 100,000 which contain '123':
  1123   1231   1237   8123  11239  12301  12323  12329  12343  12347
 12373  12377  12379  12391  17123  20123  22123  28123  29123  31123
 31231  31237  34123  37123  40123  41231  41233  44123  47123  49123
 50123  51239  56123  59123  61231  64123  65123  70123  71233  71237
 76123  81233  81239  89123  91237  98123

Found      4 such primes < 10^4
Found     46 such primes < 10^5
Found    451 such primes < 10^6
Found  4,412 such primes < 10^7
Found 43,548 such primes < 10^8
```

Prime triplets

```
say "Values of p such that (p, p+2, p+6) are all prime:"
5500.primes.grep{|p| all_prime(p+2, p+6) }.say
```

Output:

```
Values of p such that (p, p+2, p+6) are all prime:
[5, 11, 17, 41, 101, 107, 191, 227, 311, 347, 461, 641, 821, 857, 881, 1091, 1277, 1301, 1427, 1481, 14
```

Primes - allocate descendants to their ancestors

```
var maxsum = 99
var primes = maxsum.primes

var descendants = (maxsum+1).of { [] }
var ancestors   = (maxsum+1).of { [] }

for p in (primes) {
    descendants[p] << p
    for s in (1 .. descendants.end-p) {
        descendants[s + p] << descendants[s].map {|q| p*q }...
    }
}

for p in (primes + [4]) {
    descendants[p].pop
}

var total = 0

for s in (1 .. maxsum) {

    descendants[s].sort!

    total += (var dsclen = descendants[s].len)
    var idx = descendants[s].first_index {|x| x > maxsum }

    for d in (descendants[s].slice(0, idx)) {
        ancestors[d] = (ancestors[s] + [s])
    }

    if ((s <= 20) || (s ~~ [46, 74, 99])) {
        printf("%2d: %d Ancestor(s): %-15s %5s Descendant(s): %s\n", s,
            ancestors[s].len, "[#{ancestors[s].join(' ')}]", descendants[s].len,
            dsclen <= 10 ? descendants[s] : "[#{descendants[s].first(10).join(' ')} ...]")
    }
}

say "\nTotal descendants: #{total}"
```

Output:

```
 1: 0 Ancestor(s): []                  0 Descendant(s): []
 2: 0 Ancestor(s): []                  0 Descendant(s): []
 3: 0 Ancestor(s): []                  0 Descendant(s): []
 4: 0 Ancestor(s): []                  0 Descendant(s): []
 5: 0 Ancestor(s): []                  1 Descendant(s): [6]
 6: 1 Ancestor(s): [5]                 2 Descendant(s): [8, 9]
 7: 0 Ancestor(s): []                  2 Descendant(s): [10, 12]
 8: 2 Ancestor(s): [5 6]               3 Descendant(s): [15, 16, 18]
 9: 2 Ancestor(s): [5 6]               4 Descendant(s): [14, 20, 24, 27]
10: 1 Ancestor(s): [7]                 5 Descendant(s): [21, 25, 30, 32, 36]
11: 0 Ancestor(s): []                  5 Descendant(s): [28, 40, 45, 48, 54]
12: 1 Ancestor(s): [7]                 7 Descendant(s): [35, 42, 50, 60, 64, 72, 81]
13: 0 Ancestor(s): []                  8 Descendant(s): [22, 56, 63, 75, 80, 90, 96, 108]
14: 3 Ancestor(s): [5 6 9]            10 Descendant(s): [33, 49, 70, 84, 100, 120, 128, 135, 144, 162]
15: 3 Ancestor(s): [5 6 8]            12 Descendant(s): [26 44 105 112 125 126 150 160 180 192 ...]
16: 3 Ancestor(s): [5 6 8]            14 Descendant(s): [39 55 66 98 140 168 189 200 225 240 ...]
17: 0 Ancestor(s): []                 16 Descendant(s): [52 88 99 147 175 210 224 250 252 300 ...]
18: 3 Ancestor(s): [5 6 8]            19 Descendant(s): [65 77 78 110 132 196 280 315 336 375 ...]
19: 0 Ancestor(s): []                 22 Descendant(s): [34 104 117 165 176 198 245 294 350 420 ...]
20: 3 Ancestor(s): [5 6 9]            26 Descendant(s): [51 91 130 154 156 220 264 297 392 441 ...]
46: 3 Ancestor(s): [7 10 25]         557 Descendant(s): [129 205 246 493 518 529 740 806 888 999 ...]
74: 5 Ancestor(s): [5 6 8 16 39]    6336 Descendant(s): [213 469 670 793 804 1333 1342 1369 1534 2014 .
99: 1 Ancestor(s): [17]            38257 Descendant(s): [194 1869 2225 2670 2848 3204 3237 4029 4565 50

Total descendants: 546986
```

Primes which contain only one odd digit

```
func primes_with_one_odd_digit(upto, base = 10) {

    var list = []
    var digits = @(^base)

    var even_digits = digits.grep { .is_even }
    var odd_digits  = digits.grep { .is_odd && .is_coprime(base) }

    list << digits.grep { .is_odd && .is_prime && !.is_coprime(base) }...

    for k in (0 .. upto.len(base)-1) {
        even_digits.variations_with_repetition(k, {|*a|
            next if (a.last == 0)
            break if ([1, a...].digits2num(base) > upto)
            odd_digits.each {|d|
                var n = [d, a...].digits2num(base)
                list << n if n.is_prime
            }
        })
    }

    list.sort
}

with (1e3) {|n|
    var list = primes_with_one_odd_digit(n)
    say "There are #{list.len} primes under #{n.commify} which contain only one odd digit:"
    list.each_slice(9, {|*a| say a.map { '%3s' % _ }.join(' ') })
}

say ''

for k in (1..8) {
    var count = primes_with_one_odd_digit(10**k).len
    say "There are #{'%6s' % count.commify} such primes <= 10^#{k}"
}
```

Output:

```
There are 45 primes under 1,000 which contain only one odd digit:
  3   5   7  23  29  41  43  47  61
 67  83  89 223 227 229 241 263 269
281 283 401 409 421 443 449 461 463
467 487 601 607 641 643 647 661 683
809 821 823 827 829 863 881 883 887

There are      3 such primes <= 10^1
There are     12 such primes <= 10^2
There are     45 such primes <= 10^3
There are    171 such primes <= 10^4
There are    619 such primes <= 10^5
There are  2,560 such primes <= 10^6
There are 10,774 such primes <= 10^7
There are 46,708 such primes <= 10^8
```

Primes whose first and last number is 3

```
func numbers_with_edges(upto, base = 10, s = [3]) {
    Enumerator({|callback|
        callback(s.digits2num(base))
        for k in (0 .. base**(upto.len(base) - 2*s.len)) {

            break if (s + k.digits(base) + s -> digits2num(base) > upto)

            Inf.times { |j|

                var d = (s + k.digits(base) + j.of(0) + s)
                var n = d.digits2num(base)

                (n <= upto) ? callback(n) : break
            }
        }
    })
}

with (4e3) { |n|
    var list  = numbers_with_edges(n).grep{.is_prime}.sort
    say "There are #{list.len} primes <= #{n.commify} which begin and end in 3:"
    list.each_slice(10, {|*a| say a.map { '%5s' % _ }.join(' ') })
}

with (1e6) {|n|
    var count = numbers_with_edges(n).grep{.is_prime}.len
    say "\nThere are #{count} primes <= #{n.commify} which begin and end in 3"
}
```

Output:

```
There are 33 primes <= 4,000 which begin and end in 3:
    3   313   353   373   383  3023  3083  3163  3203  3253
 3313  3323  3343  3373  3413  3433  3463  3533  3583  3593
 3613  3623  3643  3673  3733  3793  3803  3823  3833  3853
 3863  3923  3943

There are 2251 primes <= 1,000,000 which begin and end in 3
```

Primes whose sum of digits is 25

Simple solution:

```
5000.primes.grep { .sumdigits == 25 }.say
```

Generate such primes from digits (asymptotically faster):

```
func generate_from_prefix(limit, digitsum, p, base, digits, t=p) {

    var seq = [p]

    digits.each {|d|
        var num = (p*base + d)
        num <= limit    || return seq

        var sum = (t + d)
        sum <= digitsum || return seq

        seq << __FUNC__(limit, digitsum, num, base, digits, sum)\
               .grep { .is_prime }...
    }

    return seq
}

func primes_with_digit_sum(limit, digitsum = 25, base = 10, digits = @(^base)) {
    digits.grep { _ > 0 }\
          .map  { generate_from_prefix(limit, digitsum, _, base, digits)... }\
          .grep { .sumdigits(base) == digitsum }\
          .sort
}

say primes_with_digit_sum(5000)
```

Output:

```
[997, 1699, 1789, 1879, 1987, 2689, 2797, 2887, 3499, 3697, 3769, 3877, 3967, 4597, 4759, 4957, 4993]
```

Primes with digits in nondecreasing order

Simple solution:

```
say 1000.primes.grep { .digits.cons(2).all { .head >= .tail } }
```

Generate such primes from digits (asymptotically faster):

```
func primes_with_nondecreasing_digits(upto, base = 10) {

    upto = prev_prime(upto+1)

    var list = []
    var digits = @(1..^base -> flip)

    var end_digits = digits.grep { .is_coprime(base) }
    list << digits.grep { .is_prime && !.is_coprime(base) }...

    for k in (0 .. upto.ilog(base)) {
        digits.combinations_with_repetition(k, {|*a|
            var v = a.digits2num(base)
            end_digits.each {|d|
                var n = (v*base + d)
                next if ((n >= base) && (a[0] > d))
                list << n if (n.is_prime && (n <= upto))
            }
        })
    }

    list.sort
}

say primes_with_nondecreasing_digits(1000)
```

Output:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 37, 47, 59, 67, 79, 89, 113, 127, 137, 139, 149, 157, 167, 179, 19
```

Primorial numbers

```
say (
    'First ten primorials: ',
    {|i| pn_primorial(i) }.map(^10).join(', ')
)

{ |i|
    say ("primorial(10^#{i}) has " + pn_primorial(10**i).len + ' digits')
} << 1..6
```

Output:

```
First ten primorials: 1, 2, 6, 30, 210, 2310, 30030, 510510, 9699690, 223092870
primorial(10^1) has 10 digits
primorial(10^2) has 220 digits
primorial(10^3) has 3393 digits
primorial(10^4) has 45337 digits
primorial(10^5) has 563921 digits
primorial(10^6) has 6722809 digits
```

Priority queue

```
class PriorityQueue {
    has tasks = []

    method insert (Number priority { _ >= 0 }, task) {
        for n in range(tasks.len, priority) {
            tasks[n] = []
        }
        tasks[priority].append(task)
    }

    method get      { tasks.first { !.is_empty } -> shift }
    method is_empty { tasks.all   {  .is_empty } }
}

var pq = PriorityQueue()

[
    [3, 'Clear drains'],
    [4, 'Feed cat'],
    [5, 'Make tea'],
    [9, 'Sleep'],
    [3, 'Check email'],
    [1, 'Solve RC tasks'],
    [9, 'Exercise'],
    [2, 'Do taxes'],
].each { |pair|
    pq.insert(pair...)
}

say pq.get while !pq.is_empty
```

Output:

```
Solve RC tasks
Do taxes
Clear drains
Check email
Feed cat
Make tea
Sleep
Exercise
```

Probabilistic choice

```
define TRIALS = 1e4

func prob_choice_picker(options) {
    var n = 0
    var a = []
    options.each { |k,v|
        n += v
        a << [n, k]
    }
    func {
        var r = 1.rand
        a.first{|e| r <= e[0] }[1]
    }
}

var ps = Hash(
   aleph  => 1/5,
   beth   => 1/6,
   gimel  => 1/7,
   daleth => 1/8,
   he     => 1/9,
   waw    => 1/10,
   zayin  => 1/11
)

ps{:heth} = (1 - ps.values.sum)

var picker = prob_choice_picker(ps)
var results = Hash()

TRIALS.times {
    results{picker()} := 0 ++
}

say "Event   Occurred  Expected  Difference"
for k,v in (results.sort_by {|k| results{k} }.reverse) {
    printf("%-6s  %f  %f  %f\n",
        k, v/TRIALS, ps{k},
        abs(v/TRIALS - ps{k})
    )
}
```

Output:

```
Event   Occurred  Expected  Difference
aleph   0.196300  0.200000  0.003700
beth    0.165600  0.166667  0.001067
gimel   0.143700  0.142857  0.000843
daleth  0.123900  0.125000  0.001100
he      0.111800  0.111111  0.000689
waw     0.101900  0.100000  0.001900
zayin   0.088100  0.090909  0.002809
heth    0.068800  0.063456  0.005344
```

Problem of Apollonius

```
class Circle(x,y,r) {
    method to_s { "Circle(#{x}, #{y}, #{r})" }
}

func solve_apollonius(c, s) {

    var(c1, c2, c3) = c...
    var(s1, s2, s3) = s...

    var v11 = (2*c2.x - 2*c1.x)
    var v12 = (2*c2.y - 2*c1.y)
    var v13 = (c1.x**2 - c2.x**2 + c1.y**2 - c2.y**2 - c1.r**2 + c2.r**2)
    var v14 = (2*s2*c2.r - 2*s1*c1.r)

    var v21 = (2*c3.x - 2*c2.x)
    var v22 = (2*c3.y - 2*c2.y)
    var v23 = (c2.x**2 - c3.x**2 + c2.y**2 - c3.y**2 - c2.r**2 + c3.r**2)
    var v24 = (2*s3*c3.r - 2*s2*c2.r)

    var w12 = (v12 / v11)
    var w13 = (v13 / v11)
    var w14 = (v14 / v11)

    var w22 = (v22/v21 - w12)
    var w23 = (v23/v21 - w13)
    var w24 = (v24/v21 - w14)

    var P = (-w23 / w22)
    var Q = (w24 / w22)
    var M = (-w12*P - w13)
    var N = (w14 - w12*Q)

    var a = (N**2 + Q**2 - 1)
    var b = (2*M*N - 2*N*c1.x + 2*P*Q - 2*Q*c1.y + 2*s1*c1.r)
    var c = (c1.x**2 + M**2 - 2*M*c1.x + P**2 + c1.y**2 - 2*P*c1.y - c1.r**2)

    var D = (b**2 - 4*a*c)
    var rs = ((-b - D.sqrt) / 2*a)

    var xs = (M + N*rs)
    var ys = (P + Q*rs)

    Circle(xs, ys, rs)
}

var c = [Circle(0, 0, 1), Circle(4, 0, 1), Circle(2, 4, 2)]
say solve_apollonius(c, %n<1 1 1>)
say solve_apollonius(c, %n<-1 -1 -1>)
```

Output:

```
Circle(2, 2.1, 3.9)
Circle(2, 0.83333333333333333333333333333333333333333325, 1.16666666666666666666666666666666666666666667)
```

Program name

```
say __MAIN__
if (__MAIN__ != __FILE__) {
    say "This file has been included!"
}
```

Program termination

```
if (problem) {
    Sys.exit(code)
}
```

Proper divisors

```
func propdiv (n) {
    n.divisors.slice(0, -2)
}

{|i| printf("%2d: %s\n", i, propdiv(i)) } << 1..10

var max = 0
var candidates = []

for i in (1..20_000) {
    var divs = propdiv(i).len
    if (divs > max) {
        candidates = []
        max = divs
    }
    candidates << i if (divs == max)
}

say "max = #{max}, candidates = #{candidates}"
```

Output:

```
 1: []
 2: [1]
 3: [1]
 4: [1, 2]
 5: [1]
 6: [1, 2, 3]
 7: [1]
 8: [1, 2, 4]
 9: [1, 3]
10: [1, 2, 5]
max = 79, candidates = [15120, 18480]
```

Pseudo-random numbers/Combined recursive generator MRG32k3a

```
class MRG32k3a(seed) {

    define(
        m1 = (2**32 - 209)
        m2 = (2**32 - 22853)
    )

    define(
        a1 = %n<     0 1403580  -810728>
        a2 = %n<527612       0 -1370589>
    )

    has x1 = [seed, 0, 0]
    has x2 = x1.clone

    method next_int {
        x1.unshift(a1.map_kv {|k,v| v * x1[k] }.sum % m1); x1.pop
        x2.unshift(a2.map_kv {|k,v| v * x2[k] }.sum % m2); x2.pop
        (x1[0] - x2[0]) % (m1 + 1)
    }

    method next_float { self.next_int / (m1 + 1) -> float }
}

say "Seed: 1234567, first 5 values:"
var rng = MRG32k3a(seed: 1234567)
5.of { rng.next_int }.each { .say }

say "\nSeed: 987654321, values histogram:";
var rng = MRG32k3a(seed: 987654321)
var freq = 100_000.of { rng.next_float * 5 -> int }.freq
freq.sort.each_2d {|k,v| say "#{k} #{v}" }
```

Output:

```
Seed: 1234567, first 5 values:
1459213977
2827710106
4245671317
3877608661
2595287583

Seed: 987654321, values histogram:
0 20002
1 20060
2 19948
3 20059
4 19931
```

Pseudo-random numbers/Middle-square method

```
class MiddleSquareMethod(seed, k = 1000) {
    method next {
        seed = (seed**2 // k % k**2)
    }
}

var obj = MiddleSquareMethod(675248)
say 5.of { obj.next }
```

Output:

```
[959861, 333139, 981593, 524817, 432883]
```

Pseudo-random numbers/PCG32

```
class PCG32(seed, incr) {

    has state

    define (
        mask32 = (2**32 - 1),
        mask64 = (2**64 - 1),
        N      = 6364136223846793005,
    )

    method init {
        seed := 1
        incr := 2
        incr  = (((incr << 1) | 1) & mask64)
        state = (((incr + seed)*N + incr) & mask64)
    }

    method next_int {
        var shift  = ((((state >> 18) ^ state) >> 27) & mask32)
        var rotate = ((state >> 59) & mask32)
            state  = ((state*N + incr) & mask64)
        ((shift >> rotate) | (shift << (32-rotate))) & mask32
    }

    method next_float {
        self.next_int / (mask32+1)
    }
}

say "Seed: 42, Increment: 54, first 5 values:";
var rng = PCG32(seed: 42, incr: 54)
say 5.of { rng.next_int }

say "\nSeed: 987654321, Increment: 1, values histogram:";
var rng = PCG32(seed: 987654321, incr: 1)
var histogram = Bag(1e5.of { floor(5*rng.next_float) }...)
histogram.pairs.sort.each { .join(": ").say }
```

Output:

```
Seed: 42, Increment: 54, first 5 values:
[2707161783, 2068313097, 3122475824, 2211639955, 3215226955]

Seed: 987654321, Increment: 1, values histogram:
0: 20049
1: 20022
2: 20115
3: 19809
4: 20005
```

Pseudo-random numbers/Splitmix64

```
class Splitmix64(state) {

    define (
        mask64 = (2**64 - 1)
    )

    method next_int {
        var n = (state = ((state + 0x9e3779b97f4a7c15) & mask64))
        n = ((n ^ (n >> 30)) * 0xbf58476d1ce4e5b9 & mask64)
        n = ((n ^ (n >> 27)) * 0x94d049bb133111eb & mask64)
        (n ^ (n >> 31)) & mask64
    }

    method next_float {
        self.next_int / (mask64+1)
    }
}

say 'Seed: 1234567, first 5 values:'
var rng = Splitmix64(1234567)
5.of { rng.next_int.say }

say "\nSeed: 987654321, values histogram:"
var rng = Splitmix64(987654321)
var histogram = Bag(1e5.of { floor(5*rng.next_float) }...)
histogram.pairs.sort.each { .join(": ").say }
```

Output:

```
Seed: 1234567, first 5 values:
6457827717110365317
3203168211198807973
9817491932198370423
4593380528125082431
1640892285945822 3821

Seed: 987654321, values histogram:
0: 20027
1: 19892
2: 20073
3: 19978
4: 20030
```

Pseudo-random numbers/Xorshift star

```
class Xorshift_star(state) {

    define (
        mask32 = (2**32 - 1),
        mask64 = (2**64 - 1),
    )

    method next_int {
        state ^= (state >> 12)
        state ^= (state << 25 & mask64)
        state ^= (state >> 27)
        (state * 0x2545F4914F6CDD1D) >> 32 & mask32
    }

    method next_float {
        self.next_int / (mask32+1)
    }
}

say 'Seed: 1234567, first 5 values:';
var rng = Xorshift_star(1234567)
say 5.of { rng.next_int }

say "\nSeed: 987654321, values histogram:";
var rng = Xorshift_star(987654321)
var histogram = Bag(1e5.of { floor(5*rng.next_float) }...)
histogram.pairs.sort.each { .join(": ").say }
```

Output:

```
Seed: 1234567, first 5 values:
[3540625527, 2750739987, 4037983143, 1993361440, 3809424708]

Seed: 987654321, values histogram:
0: 20103
1: 19922
2: 19937
3: 20031
4: 20007
```

Pseudorandom number generator image

```
require('GD')

var img = %O<GD::Image>.new(500, 500, 1)

for y in (0..500), x in (0..500) {
    var color = img.colorAllocate(255.irand, 255.irand, 255.irand)
    img.setPixel(x, y, color)
}

File("image500.png").write(img.png, :raw)
```

Pythagoras tree

```
require('Imager')

func tree(img, x1, y1, x2, y2, depth) {

    depth <= 0 && return()

    var dx = (x2 - x1)
    var dy = (y1 - y2)

    var x3 = (x2 - dy)
    var y3 = (y2 - dx)
    var x4 = (x1 - dy)
    var y4 = (y1 - dx)
    var x5 = (x4 + 0.5*(dx - dy))
    var y5 = (y4 - 0.5*(dx + dy))

    # square
    img.polygon(
        points => [
            [x1, y1],
            [x2, y2],
            [x3, y3],
            [x4, y4],
        ],
        color => [0, 255/depth, 0],
    )

    # triangle
    img.polygon(
        points => [
            [x3, y3],
            [x4, y4],
            [x5, y5],
        ],
        color => [0, 255/depth, 0],
    )

    tree(img, x4, y4, x5, y5, depth - 1)
    tree(img, x5, y5, x3, y3, depth - 1)
}

var (width=1920, height=1080)
var img = %O<Imager>.new(xsize => width, ysize => height)
img.box(filled => 1, color => 'white')
tree(img, width/2.3, height, width/1.8, height, 10)
img.write(file => 'pythagoras_tree.png')
```

Output image

Pythagorean quadruples

```
# Finds all solutions (a,b) such that: a^2 + b^2 = n^2
func sum_of_two_squares(n) is cached {

    n == 0 && return [[0, 0]]

    var prod1 = 1
    var prod2 = 1

    var prime_powers = []
```

```
    for p,e in (n.factor_exp) {
        if (p % 4 == 3) {                   # p = 3 (mod 4)
            e.is_even || return []          # power must be even
            prod2 *= p**(e >> 1)
        }
        elsif (p == 2) {                    # p = 2
            if (e.is_even) {                # power is even
                prod2 *= p**(e >> 1)
            }
            else {                          # power is odd
                prod1 *= p
                prod2 *= p**((e - 1) >> 1)
                prime_powers.append([p, 1])
            }
        }
        else {                              # p = 1 (mod 4)
            prod1 *= p**e
            prime_powers.append([p, e])
        }
    }

    prod1 == 1 && return [[prod2, 0]]
    prod1 == 2 && return [[prod2, prod2]]

    # All the solutions to the congruence: x^2 = -1 (mod prod1)
    var square_roots = gather {
        gather {
            for p,e in (prime_powers) {
                var pp = p**e
                var r = sqrtmod(-1, pp)
                take([[r, pp], [pp - r, pp]])
            }
        }.cartesian { |*a|
            take(Math.chinese(a...))
        }
    }

    var solutions = []

    for r in (square_roots) {

        var s = r
        var q = prod1

        while (s*s > prod1) {
            (s, q) = (q % s, s)
        }

        solutions.append([prod2 * s, prod2 * (q % s)])
    }

    for p,e in (prime_powers) {
        for (var i = e%2; i < e; i += 2) {

            var sq = p**((e - i) >> 1)
            var pp = p**(e - i)

            solutions += (
                __FUNC__(prod1 / pp).map { |pair|
                    pair.map {|r| sq * prod2 * r }
                }
            )
        }
    }
```

```
    solutions.map     {|pair| pair.sort } \
             .uniq_by {|pair| pair[0]   } \
             .sort_by {|pair| pair[0]   }
}

# Finds solutions (a,b,c) such that: a^2 + b^2 + c^2 = n^2
func sum_of_three_squares(n) {
    gather {
        for k in (1 .. n//3) {
            var t = sum_of_two_squares(n**2 - k**2) || next
            take(t.map { [k, _...] }...)
        }
    }
}

say gather {
    for n in (1..2200) {
        sum_of_three_squares(n) || take(n)
    }
}
```

Output:

```
[1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 128, 160, 256, 320, 512, 640, 1024, 1280, 2048]
```

Numbers d that cannot be expressed as a^2 + b^2 + c^2 = d^2, are numbers of the form 2^n or 5*2^n:

```
say gather {
    for n in (1..2200) {
        if ((n & (n-1) == 0) || (n%%5 && ((n/5) & (n/5 - 1) == 0))) {
            take(n)
        }
    }
}
```

Output:

```
[1, 2, 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 128, 160, 256, 320, 512, 640, 1024, 1280, 2048]
```

Pythagorean triples

```
func triples(limit) {
    var primitive = 0
    var civilized = 0

    func oyako(a, b, c) {
        (var perim = a+b+c) > limit || (
            primitive++
            civilized += int(limit / perim)
            oyako( a - 2*b + 2*c,  2*a - b + 2*c,  2*a - 2*b + 3*c)
            oyako( a + 2*b + 2*c,  2*a + b + 2*c,  2*a + 2*b + 3*c)
            oyako(-a + 2*b + 2*c, -2*a + b + 2*c, -2*a + 2*b + 3*c)
        )
    }

    oyako(3,4,5)
    "#{limit} => (#{primitive} #{civilized})"
}

for n (1..Inf) {
    say triples(10**n)
}
```

Output:

```
10 => (0 0)
100 => (7 17)
1000 => (70 325)
10000 => (703 4858)
100000 => (7026 64741)
^C
```

Quadrat special primes

```
func quadrat_primes(callback) {

    var prev = 2
    callback(prev)

    loop {
        var curr = (1..Inf -> lazy.map { prev + _**2 }.first { .is_prime })
        callback(curr)
        prev = curr
    }
}

say gather {
    quadrat_primes({|k|
        break if (k >= 16000)
        take(k)
    })
}
```

Output:

```
[2, 3, 7, 11, 47, 83, 227, 263, 587, 911, 947, 983, 1019, 1163, 1307, 1451, 1487, 1523, 1559, 2459, 335
```

Quaternion type

```
class Quaternion(r, i, j, k) {

    func qu(*r) { Quaternion(r...) }

    method to_s  { "#{r} + #{i}i + #{j}j + #{k}k" }
    method reals { [r, i, j, k] }
    method conj  { qu(r, -i, -j, -k) }
    method norm  { self.reals.map { _*_ }.sum.sqrt }

    method ==(Quaternion b) { self.reals == b.reals }

    method +(Number     b) { qu(b+r, i, j, k) }
    method +(Quaternion b) { qu((self.reals ~Z+ b.reals)...) }

    method neg { qu(self.reals.map{ .neg }...) }

    method *(Number     b) { qu((self.reals»*»b)...) }
    method *(Quaternion b) {
        var (r,i,j,k) = b.reals...
        qu(sum(self.reals ~Z* [r, -i, -j, -k]),
           sum(self.reals ~Z* [i,  r,  k, -j]),
           sum(self.reals ~Z* [j, -k,  r,  i]),
           sum(self.reals ~Z* [k,  j, -i,  r]))
    }
}

var q  = Quaternion(1, 2, 3, 4)
var q1 = Quaternion(2, 3, 4, 5)
var q2 = Quaternion(3, 4, 5, 6)
var r  = 7

say "1) q norm  = #{q.norm}"
say "2) -q      = #{-q}"
say "3) q conj  = #{q.conj}"
say "4) q  + r  = #{q + r}"
say "5) q1 + q2 = #{q1 + q2}"
say "6) q  * r  = #{q  * r}"
say "7) q1 * q2 = #{q1 * q2}"
say "8) q1q2 #{ q1*q2 == q2*q1 ? '==' : '!=' } q2q1"
```

Output:

```
1) q norm  = 5.477225575051661134569697828008021339527446949 98
2) -q      = -1 + -2i + -3j + -4k
3) q conj  = 1 + -2i + -3j + -4k
4) q  + r  = 8 + 2i + 3j + 4k
5) q1 + q2 = 5 + 7i + 9j + 11k
6) q  * r  = 7 + 14i + 21j + 28k
7) q1 * q2 = -56 + 16i + 24j + 26k
8) q1q2 != q2q1
```

Queue/Definition

Implemented as a class:

```
class FIFO(*array) {
    method pop {
        array.is_empty && die "underflow"
        array.shift
    }
    method push(*items) {
        array += items
        self
    }
    method empty {
        array.len == 0
    }
}
```

Queue/Usage

```
var f = FIFO()
say f.empty         # true
f.push('foo')
f.push('bar', 'baz')
say f.pop           # foo
say f.empty         # false

var g = FIFO('xxx', 'yyy')
say g.pop           # xxx
say f.pop           # bar
```

Quickselect algorithm

```
func quickselect(a, k) {
    var pivot = a.pick
    var left  = a.grep{|i| i < pivot}
    var right = a.grep{|i| i > pivot}

    given(left.len) { |l|
        when(k)     { pivot }
        case(k < l) { __FUNC__(left, k) }
        default     { __FUNC__(right, k - l - 1) }
    }
}

var v = [9, 8, 7, 6, 5, 0, 1, 2, 3, 4]
say v.range.map{|i| quickselect(v, i)}
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Quine

With printf():

```
s = %(s = %%(%s); printf(s, s);
); printf(s, s);
```

With HERE-doc:

```
say(<<e*2, 'e')
say(<<e*2, 'e')
e
```

Ramanujan's constant

```
func ramanujan_const(x, decimals=32) {
    local Num!PREC = *"#{4*round((Num.pi*√x)/log(10) + decimals + 1)}"
    exp(Num.pi * √x) -> round(-decimals).to_s
}

var decimals = 100
printf("Ramanujan's constant to #{decimals} decimals:\n%s\n\n",
     ramanujan_const(163, decimals))

say "Heegner numbers yielding 'almost' integers:"
[19, 96, 43, 960, 67, 5280, 163, 640320].each_slice(2, {|h,x|
    var c = ramanujan_const(h, 32)
    var n = (x**3 + 744)
    printf("%3s: %51s ≈ %18s (diff: %s)\n", h, c, n, n-Num(c))
})
```

Output:

```
Ramanujan's constant to 100 decimals:
262537412640768743.99999999999925007259719818568887935385633733699086270753741037821064791011860731295 1

Heegner numbers yielding 'almost' integers:
 19:             885479.77768015431949753789348171962682 ≈             885480 (diff: 0.2223198456805024)
 43:          884736743.99977746603490666193746207858538 ≈          884736744 (diff: 0.0002225339650933)
 67:       147197952743.99999866245422450682926131257863 ≈       147197952744 (diff: 0.0000013375457754)
163: 262537412640768743.99999999999925007259719818568888 ≈ 262537412640768744 (diff: 0.0000000000007499)
```

Ramer-Douglas-Peucker line simplification

```
func perpendicular_distance(Arr start, Arr end, Arr point) {
    ((point == start) || (point == end)) && return 0
    var (Δx,  Δy ) = (  end »-« start)...
    var (Δpx, Δpy) = (point »-« start)...
    var h = hypot(Δx, Δy)
    [\Δx, \Δy].map { *_ /= h }
    (([Δpx, Δpy] »-« ([Δx, Δy] »*» (Δx*Δpx + Δy*Δpy))) »**» 2).sum.sqrt.round(-20)
}

func Ramer_Douglas_Peucker(Arr points { .all { .len > 1 } }, ε = 1) {
    points.len == 2 && return points

    var d = (^points -> map {
        perpendicular_distance(points[0], points[-1], points[_])
    })

    if (d.max > ε) {
        var i = d.index(d.max)
        return [Ramer_Douglas_Peucker(points.ft(0, i), ε).ft(0, -2)...,
                Ramer_Douglas_Peucker(points.ft(i),    ε)...]
    }

    return [points[0,-1]]
}

say Ramer_Douglas_Peucker(
    [[0,0],[1,0.1],[2,-0.1],[3,5],[4,6],[5,7],[6,8.1],[7,9],[8,9],[9,9]]
)
```

Output:

```
[[0, 0], [2, -1/10], [3, 5], [7, 9], [9, 9]]
```

Ramsey's theorem

```
var a = 17.of { 17.of(0) }

17.times {|i| a[i][i] = '-' }
4.times { |k|
  17.times { |i|
    var j = ((i + 1<<k) % 17)
    a[i][j] = (a[j][i] = 1)
  }
}

a.each {|row| say row.join(' ') }

combinations(17, 4, { |*quartet|
  var links = quartet.combinations(2).map{|p| a.dig(p...) }.sum
  ((0 < links) && (links < 6)) || die "Bogus!"
})
say "Ok"
```

Output:

```
- 1 1 0 1 0 0 0 1 1 0 0 0 1 0 1 1
1 - 1 1 0 1 0 0 0 1 1 0 0 0 1 0 1
1 1 - 1 1 0 1 0 0 0 1 1 0 0 0 1 0
0 1 1 - 1 1 0 1 0 0 0 1 1 0 0 0 1
1 0 1 1 - 1 1 0 1 0 0 0 1 1 0 0 0
0 1 0 1 1 - 1 1 0 1 0 0 0 1 1 0 0
0 0 1 0 1 1 - 1 1 0 1 0 0 0 1 1 0
0 0 0 1 0 1 1 - 1 1 0 1 0 0 0 1 1
1 0 0 0 1 0 1 1 - 1 1 0 1 0 0 0 1
1 1 0 0 0 1 0 1 1 - 1 1 0 1 0 0 0
0 1 1 0 0 0 1 0 1 1 - 1 1 0 1 0 0
0 0 1 1 0 0 0 1 0 1 1 - 1 1 0 1 0
0 0 0 1 1 0 0 0 1 0 1 1 - 1 1 0 1
1 0 0 0 1 1 0 0 0 1 0 1 1 - 1 1 0
0 1 0 0 0 1 1 0 0 0 1 0 1 1 - 1 1
1 0 1 0 0 0 1 1 0 0 0 1 0 1 1 - 1
1 1 0 1 0 0 0 1 1 0 0 0 1 0 1 1 -
Ok
```

Random number generator (device)

```
func urandom() {
    static device = %f'/dev/urandom'

    device.open('<:raw', \var fh, \var err) ->
        || die "Can't open `#{device}': #{err}"

    fh.sysread(\var noise, 4)
    'L'.unpack(noise)
}

say urandom()    # sample: 3517432564
```

Random number generator (included)

Latest versions of Sidef use the Mersenne Twister algorithm to compute pseudorandom numbers, with different initial seeds (and implementations) for floating-points and integers.

```
say 1.rand          # random float in the interval [0,1)
say 100.irand       # random integer in the interval [0,100]
```

Random numbers

```
var arr = 1000.of { 1 + (0.5 * sqrt(-2 * 1.rand.log) * cos(Num.tau * 1.rand)) }
arr.each { .say }
```

Range expansion

```
func rangex(str) {
    str.split(',').map { |r|
        var m = r.match(/^
            (?(DEFINE) (?<int>[+-]?[0-9]+) )
            (?<from>(?&int))-(?<to>(?&int))
        $/x)
        m ? do {var c = m.ncap; @(Num(c{:from}) .. Num(c{:to}))}
          : Num(r)
    }
}

say rangex('-6,-3--1,3-5,7-11,14,15,17-20').flatten.join(',')
```

Output:

```
-6,-3,-2,-1,3,4,5,7,8,9,10,11,14,15,17,18,19,20
```

Ranking methods

```
var scores = [
    Pair(Solomon => 44),
    Pair(Jason   => 42),
    Pair(Errol   => 42),
    Pair(Garry   => 41),
    Pair(Bernard => 41),
    Pair(Barry   => 41),
    Pair(Stephen => 39),
]

func tiers(s) {
    s.group_by { .value }.kv.sort.flip.map { .value.map{.key} }
}

func standard(s) {
    var rank = 1
    gather {
        for players in tiers(s) {
            take(Pair(rank, players))
            rank += players.len
        }
    }
}

func modified(s) {
    var rank = 0
    gather {
        for players in tiers(s) {
            rank += players.len
            take(Pair(rank, players))
        }
    }
}

func dense(s) {
    tiers(s).map_kv { |k,v| Pair(k+1, v) }
}

func ordinal(s) {
    s.map_kv { |k,v| Pair(k+1, v.key) }
}

func fractional(s) {
    var rank = 1
    gather {
        for players in tiers(s) {
            var beg = rank
            var end = (rank += players.len)
            take(Pair(sum(beg ..^ end) / players.len, players))
        }
    }
}

func display(r) {
    say r.map {|a| '%3s : %s' % a... }.join("\n")
}

say   "Standard:";   display(  standard(scores))
say "\nModified:";   display(  modified(scores))
say "\nDense:";      display(     dense(scores))
say "\nOrdinal:";    display(   ordinal(scores))
say "\nFractional:"; display(fractional(scores))
```

Output:

```
Standard:
  1 : ["Solomon"]
  2 : ["Jason", "Errol"]
  4 : ["Garry", "Bernard", "Barry"]
  7 : ["Stephen"]

Modified:
  1 : ["Solomon"]
  3 : ["Jason", "Errol"]
  6 : ["Garry", "Bernard", "Barry"]
  7 : ["Stephen"]

Dense:
  1 : ["Solomon"]
  2 : ["Jason", "Errol"]
  3 : ["Garry", "Bernard", "Barry"]
  4 : ["Stephen"]

Ordinal:
  1 : Solomon
  2 : Jason
  3 : Errol
  4 : Garry
  5 : Bernard
  6 : Barry
  7 : Stephen

Fractional:
  1 : ["Solomon"]
2.5 : ["Jason", "Errol"]
  5 : ["Garry", "Bernard", "Barry"]
  7 : ["Stephen"]
```

Rate counter

```
var benchmark = frequire('Benchmark')

func job1 {
    #...job1 code...
}
func job2 {
    #...job2 code...
}

const COUNT = -1   # run for one CPU second
benchmark.timethese(COUNT, Hash('Job1' => job1, 'Job2' => job2))
```

Read a configuration file

```
var fullname = (var favouritefruit = "")
var needspeeling = (var seedsremoved = false)
var otherfamily = []

ARGF.each { |line|
    var(key, value) = line.strip.split(/\h+/, 2)...

    given(key) {
        when (nil)              { }
        when (/^([#;]|\h*$)/)   { }
        when ("FULLNAME")       { fullname = value }
        when ("FAVOURITEFRUIT") { favouritefruit = value }
        when ("NEEDSPEELING")   { needspeeling = true }
        when ("SEEDSREMOVED")   { seedsremoved = true }
        when ("OTHERFAMILY")    { otherfamily = value.split(',')»strip()» }
        default                 { say "#{key}: unknown key" }
    }
}

say "fullname       = #{fullname}"
say "favouritefruit = #{favouritefruit}"
say "needspeeling   = #{needspeeling}"
say "seedsremoved   = #{seedsremoved}"

otherfamily.each_kv {|i, name|
    say "otherfamily(#{i+1}) = #{name}"
}
```

Output:

```
fullname       = Foo Barber
favouritefruit = banana
needspeeling   = true
seedsremoved   = false
otherfamily(1) = Rhu Barber
otherfamily(2) = Harry Barber
```

Read a file character by character/UTF8

```
var file = File('input.txt')        # the input file contains: "aă€女"
var fh = file.open_r                # equivalent with: file.open('<:utf8')
fh.each_char { |char|
    printf("got character #{char} [U+%04x]\n", char.ord)
}
```

Output:

```
got character a [U+0061]
got character ă [U+0103]
got character € [U+20ac]
got character 女 [U+2f25]
```

Read a file line by line

FileHandle.each{} is lazy, allowing us to do this:

```
File(__FILE__).open_r.each { |line|
    say line
}
```

Same thing explicitly:

```
var fh = File(__FILE__).open_r
while (fh.readline(\var line)) {
    say line
}
```

Read a specific line from a file

```
func getNthLine(filename, n) {
  var file = File(filename)
  file.open_r.each { |line|
    Num($.) == n && return line
  }
  warn "file #{file} does not have #{n} lines, only #{$.}\n"
  return nil
}

var line = getNthLine("/etc/passwd", 7)
say line if defined(line)
```

Read entire file

Reading an entire file as a string, can be achieved with the *FileHandle.slurp()* method, as illustrated bellow:

```
var file = File(__FILE__)
var content = file.open_r.slurp
print content
```

Starting with version 2.30, *File.read()* can do the same:

```
var file = File(__FILE__)
var content = file.read(:utf8)
print content
```

Readline interface

```
require('Term::ReadLine')

var term = %O<Term::ReadLine>.new('Example')

term.addhistory('foo')
term.addhistory('bar')

loop {
    var cmd = term.readline("Prompt: ") \\ break

    if (cmd ~~ %w[q quit]) {
        break
    }

    say "You inserted <<#{cmd}>>"
}
```

Real constants and functions

```
Num.e     # e
Num.pi    # pi
x.sqrt    # square root
x.log     # natural logarithm
x.log10   # base 10 logarithm
x.exp     # e raised to the power of x
x.abs     # absolute value
x.floor   # floor
x.ceil    # ceiling
x**y      # exponentiation
```

Recaman's sequence

```
func recamans_generator() {

    var term = 0
    var prev = 0
    var seen = Hash()

    {
        var this = (prev - term)

        if ((this <= 0) || seen{this}) {
            this = (prev + term)
        }

        prev = this
        seen{this} = true
        term++
        this
    }
}

with (recamans_generator()) { |r|
    say ("First 15 terms of the Recaman's sequence: ", 15.of { r.run }.join(', '))
}

with (recamans_generator()) {|r|
    var seen = Hash()
    Inf.times {|i|
        var n = r.run
        if (seen{n}) {
            say "First duplicate term in the series is a(#{i}) = #{n}"
            break
        }
        seen{n} = true
    }
}

with (recamans_generator()) {|r|
    var seen = Hash()
    Inf.times {|i|
        var n = r.run
        if ((n <= 1000) && (seen{n} := true) && (seen.len == 1001)) {
            say "Terms up to a(#{i}) are needed to generate 0 to 1000"
            break
        }
    }
}
```

Output:

```
First 15 terms of the Recaman's sequence: 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9
First duplicate term in the series is a(24) = 42
Terms up to a(328002) are needed to generate 0 to 1000
```

Reduced row echelon form

```
func rref (M) {
    var (j, rows, cols) = (0, M.len, M[0].len)

    for r in (^rows) {
        j < cols || return M

        var i = r
        while (!M[i][j]) {
            ++i == rows || next
            i = r
            ++j == cols && return M
        }

        M[i, r] = M[r, i] if (r != i)
        M[r] = (M[r] »/» M[r][j])

        for n in (^rows) {
            next if (n == r)
            M[n] = (M[n] »-« (M[r] »*» M[n][j]))
        }
        ++j
    }

    return M
}

func say_it (message, array) {
    say "\n#{message}"
    for row in array {
        say row.map { |n| " %5s" % n.as_rat }.join
    }
}

var M = [
    [ # base test case
      [  1,  2,  -1,  -4 ],
      [  2,  3,  -1, -11 ],
      [ -2,  0,  -3,  22 ],
    ],
    [ # mix of number styles
      [  3,   0,  -3,    1 ],
      [ .5, 3/2,  -3,   -2 ],
      [ .2, 4/5,  -1.6, .3 ],
    ],
    [ # degenerate case
      [ 1,  2,  3,  4,  3,  1],
      [ 2,  4,  6,  2,  6,  2],
      [ 3,  6, 18,  9,  9, -6],
      [ 4,  8, 12, 10, 12,  4],
      [ 5, 10, 24, 11, 15, -4],
    ],
]

M.each { |matrix|
    say_it('Original Matrix', matrix)
    say_it('Reduced Row Echelon Form Matrix', rref(matrix))
    say ''
}
```

Output:

```
Original Matrix
     1     2    -1    -4
     2     3    -1   -11
    -2     0    -3    22

Reduced Row Echelon Form Matrix
     1     0     0    -8
     0     1     0     1
     0     0     1    -2


Original Matrix
     3     0    -3     1
   1/2   3/2    -3    -2
   1/5   4/5  -8/5  3/10

Reduced Row Echelon Form Matrix
     1     0     0 -41/2
     0     1     0 -217/6
     0     0     1 -125/6


Original Matrix
     1     2     3     4     3     1
     2     4     6     2     6     2
     3     6    18     9     9    -6
     4     8    12    10    12     4
     5    10    24    11    15    -4

Reduced Row Echelon Form Matrix
     1     2     0     0     3     4
     0     0     1     0     0    -1
     0     0     0     1     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Reflection/List methods

The super-method *Object.methods()* returns an Hash with method names as keys and *LazyMethod* objects as values. Each *LazyMethod* can be called with zero or more arguments, internally invoking the method on the object on which *.methods* was called.

```
class Example {
    method foo { }
    method bar(arg) { say "bar(#{arg})" }
}

var obj = Example()
say obj.methods.keys.sort          #=> ["bar", "call", "foo", "new"]

var meth = obj.methods.item(:bar)  # `LazyMethod` representation for `obj.bar()`
meth(123)                          # calls obj.bar()
```

Regular expressions

Simple matching:

```
var str = "I am a string"
if (str =~ /string$/) {
    print "Ends with 'string'\n"
}
```

Global matching:

```
var str = <<'EOF'
    x:Foo
    y:Bar
EOF

while (var m = str=~/(\w+):(\S+)/g) {
    say "#{m[0]} -> #{m[1]}"
}
```

Substitutions:

```
var str = "I am a string"

# Substitute something mached by a regex
str.sub!(/ a /, ' another ')   # "I am a string" => "I am another string"

# Remove something matched by a regex
str -= / \Kanother /i          # "I am another string" => "I am string"

# Global subtitution with a block
str = str.gsub(/(\w+)/, {|s1| 'x' * s1.len})  # globaly replace any word with 'xxx'

say str     # prints: 'x xx xxxxxx'
```

Remove duplicate elements

```
var ary = [1,1,2,1,'redundant',[1,2,3],[1,2,3],'redundant']
say ary.uniq
say ary.last_uniq
```

Output:

```
[1, 2, 'redundant', [1, 2, 3]]
[2, 1, [1, 2, 3], 'redundant']
```

Remove lines from a file

```
func remove_lines(file, beg, len) {
    var lines = file.open_r.lines
    lines.splice(beg, len).len == len || warn "Too few lines";
    file.write(lines.join("\n"), :utf8)
}

remove_lines(File(__FILE__), 2, 3)
```

Rename a file

```
# Here
File.rename('input.txt', 'output.txt')
File.rename('docs',      'mydocs')

# Root dir
File.rename(Dir.root + %f'input.txt', Dir.root + %f'output.txt')
File.rename(Dir.root + %f'docs',      Dir.root + %f'mydocs')
```

Rep-string

```
var arr = <1001110011 1110111011
           0010010010 1010101010
           1111111111 0100101101
           0100100  101  11 00 1>

for n (arr) {
    if (var m = /^(.+)\1+(.*$)(?(?{ substr($1, 0, length $2) eq $2 })|(?!))/.match(n)) {
       var i = m[0].len
       say (n.substr(0, i),
            n.substr(i, i).tr('01', '␀1'),
            n.substr(i*2))
    } else {
        say "#{n} (no repeat)"
    }
}
```

Output:

```
1001110011
1110111011
0010010010
1010101010
1111111111
0100101101 (no repeat)
0100100
101 (no repeat)
11
␀␀
1 (no repeat)
```

Repeat

```
func repeat(f, n) {
    { f() } * n
}

func example {
    say "Example"
}

repeat(example, 4)
```

Repeat a string

```
'ha' * 5    #==> 'hahahahaha'
```

Repunit primes

```
var limit = 1000

say "Repunit prime digits (up to #{limit}) in:"

for n in (2..20) {
    printf("Base %2d: %s\n", n,
        {|k| is_prime((n**k - 1) / (n-1)) }.grep(1..limit))
}
```

Output:

```
Base  2: [2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607]
Base  3: [3, 7, 13, 71, 103, 541]
Base  4: [2]
Base  5: [3, 7, 11, 13, 47, 127, 149, 181, 619, 929]
Base  6: [2, 3, 7, 29, 71, 127, 271, 509]
Base  7: [5, 13, 131, 149]
Base  8: [3]
Base  9: []
Base 10: [2, 19, 23, 317]
Base 11: [17, 19, 73, 139, 907]
Base 12: [2, 3, 5, 19, 97, 109, 317, 353, 701]
Base 13: [5, 7, 137, 283, 883, 991]
Base 14: [3, 7, 19, 31, 41]
Base 15: [3, 43, 73, 487]
Base 16: [2]
Base 17: [3, 5, 7, 11, 47, 71, 419]
Base 18: [2]
Base 19: [19, 31, 47, 59, 61, 107, 337]
Base 20: [3, 11, 17]
```

Resistor mesh

```
var (w, h) = (10, 10)

var v = h.of { w.of(0) } # voltage
var f = h.of { w.of(0) } # fixed condition
var d = h.of { w.of(0) } # diff
var n = []               # neighbors

for i in ^h {
    for j in (1 ..^ w  ) { n[i][j] := [] << [i, j-1] }
    for j in (0 ..^ w-1) { n[i][j] := [] << [i, j+1] }
}

for j in ^w {
    for i in (1 ..^ h  ) { n[i][j] := [] << [i-1, j] }
    for i in (0 ..^ h-1) { n[i][j] := [] << [i+1, j] }
}

func set_boundary {
    f[1][1] = 1; f[6][7] = -1;
    v[1][1] = 1; v[6][7] = -1;
}

func calc_diff {
    var total_diff = 0
    for i,j in (^h ~X ^w) {
        var w = n[i][j].map { |a| v.dig(a...) }.sum
        d[i][j] = (w = (v[i][j] - w/n[i][j].len))
        f[i][j] || (total_diff += w*w)
    }
    total_diff
}

func iter {
    var diff = 1
    while (diff > 1e-24) {
        set_boundary()
        diff = calc_diff()
        for i,j in (^h ~X ^w) {
            v[i][j] -= d[i][j]
        }
    }

    var current = 3.of(0)
    for i,j in (^h ~X ^w) {
        current[ f[i][j] ] += (d[i][j] * n[i][j].len)
    }
    (current[1] - current[-1]) / 2
}

say "R = #{2 / iter()}"
```

Output:

```
R = 1.6089912417298890219136729530730 4
```

Respond to an unknown method call

The special *AUTOLOAD* method gets called when a method isn't defined in the current class:

```
class Example {
    method foo {
        say "this is foo"
    }
    method bar {
        say "this is bar"
    }
    method AUTOLOAD(_, name, *args) {
        say ("tried to handle unknown method %s" % name)
        if (args.len > 0) {
            say ("it had arguments: %s" % args.join(', '))
        }
    }
}

var example = Example.new

example.foo           # prints “this is foo”
example.bar           # prints “this is bar”
example.grill         # prints “tried to handle unknown method grill”
example.ding("dong")  # prints “tried to handle unknown method ding”
                      # prints “it had arguments: dong”
```

Return multiple values

```
func foo(a,b) {
    return (a+b, a*b)
}
```

Catching the returned arguments:

```
var (x, y) = foo(4, 5)
say x   #=> 9
say y   #=> 20
```

Reverse a string

```
"asdf".reverse              # fdsa
"résumé niño".reverse       # oñin émusér
```

Reverse the gender of a string

```
var male2female = <<'EOD'
  maleS femaleS, maleness femaleness, him her, himself herself, his her, his
  hers, he she, Mr Mrs, Mister Missus, Ms Mr, Master Miss, MasterS MistressES,
  uncleS auntS, nephewS nieceS, sonS daughterS, grandsonS granddaughterS,
  brotherS sisterS, man woman, men women, boyS girlS, paternal maternal,
  grandfatherS grandmotherS, GodfatherS GodmotherS, GodsonS GoddaughterS,
  fiancéS fiancéeS, husband wife, husbands wives, fatherS motherS, bachelorS
  spinsterS, bridegroomS brideS, widowerS widowS, KnightS DameS, Sir DameS,
  KingS QueenS, DukeS DuchessES, PrinceS PrincessES, Lord Lady, Lords Ladies,
  MarquessES MarchionessES, EarlS CountessES, ViscountS ViscountessES, ladS
```

```
  MarquessES MarchionessES, EarlS CountessES, ViscountS ViscountessES, LadS
  lassES, sir madam, gentleman lady, gentlemen ladies, BaronS BaronessES,
  stallionS mareS, ramS eweS, coltS fillieS, billy nanny, billies nannies,
  bullS cowS, godS goddessES, heroS heroineS, shirtS blouseS, undies nickers,
  sweat glow, jackarooS jillarooS, gigoloS hookerS, landlord landlady,
  landlords landladies, manservantS maidservantS, actorS actressES, CountS
  CountessES, EmperorS EmpressES, giantS giantessES, heirS heiressES, hostS
  hostessES, lionS lionessES, managerS manageressES, murdererS murderessES,
  priestS priestessES, poetS poetessES, shepherdS shepherdessES, stewardS
  stewardessES, tigerS tigressES, waiterS waitressES, cockS henS, dogS bitchES,
  drakeS henS, dogS vixenS, tomS tibS, boarS sowS, buckS roeS, peacockS
  peahenS, gander goose, ganders geese, friarS nunS, monkS nunS
EOD

var m2f = male2female.split(/,\s*/).map { |tok| tok.words}

var re_plural = /E?S\z/
var re_ES = /ES\z/

func gen_pluralize(m, f) {
    [
        [m - re_plural, f - re_plural],
        [m.sub(re_ES, 'es'), f.sub(re_ES, 'es')],
        [m.sub(re_plural, 's'), f.sub(re_plural, 's')],
    ]
}

var dict = Hash()

for m,f in m2f {
    for x,y in gen_pluralize(m, f).map{.map{.lc}} {
        if (x ~~ dict) {
            dict{y} = x
        } else {
            dict{x, y} = (y, x)
        }
    }
}

var gen_re = Regex('\b(' + dict.keys.join('|') + ')\b', 'i')

func copy_case(orig, repl) {
    var a = orig.chars
    var b = repl.chars

    var uc = 0
    var min = [a, b].map{.len}.min
    for i in ^min {
        if (a[i] ~~ /^[[:upper:]]/) {
            b[i].uc!
            ++uc
        }
    }

    uc == min ? repl.uc : b.join('')
}

func reverse_gender(text) {
    text.gsub(gen_re, { |a| copy_case(a, dict{a.lc}) })
}
```

Example:

```
say reverse_gender("She was a soul stripper. She took my heart!")
```

Output:

```
He was a soul stripper. He took my heart!
```

Reverse words in a string

```
DATA.each {|line| line.words.reverse.join(' ').say }

__DATA__
---------- Ice and Fire ------------

fire, in end will world the say Some
ice. in say Some
desire of tasted I've what From
fire. favor who those with hold I

... elided paragraph last ...

Frost Robert -----------------------
```

Rhonda numbers

```
func is_rhonda_number(n, base = 10) {
    base.is_composite || return false
    n > 0             || return false
    n.digits(base).prod == base*n.factor.sum
}

for b in (2..16 -> grep { .is_composite }) {
    say ("First 10 Rhonda numbers to base #{b}: ",
        10.by { is_rhonda_number(_, b) })
}
```

Output:

```
First 10 Rhonda numbers to base 4: [10206, 11935, 12150, 16031, 45030, 94185, 113022, 114415, 191149, 2
First 10 Rhonda numbers to base 6: [855, 1029, 3813, 5577, 7040, 7304, 15104, 19136, 35350, 36992]
First 10 Rhonda numbers to base 8: [1836, 6318, 6622, 10530, 14500, 14739, 17655, 18550, 25398, 25956]
First 10 Rhonda numbers to base 9: [15540, 21054, 25331, 44360, 44660, 44733, 47652, 50560, 54944, 7685
First 10 Rhonda numbers to base 10: [1568, 2835, 4752, 5265, 5439, 5664, 5824, 5832, 8526, 12985]
First 10 Rhonda numbers to base 12: [560, 800, 3993, 4425, 4602, 4888, 7315, 8296, 9315, 11849]
First 10 Rhonda numbers to base 14: [11475, 18655, 20565, 29631, 31725, 45387, 58404, 58667, 59950, 639
First 10 Rhonda numbers to base 15: [2392, 2472, 11468, 15873, 17424, 18126, 19152, 20079, 24388, 30758
First 10 Rhonda numbers to base 16: [1000, 1134, 6776, 15912, 19624, 20043, 20355, 23946, 26296, 29070]
```

Rock-paper-scissors

```
const rps = %w(r p s)

const msg = [
    "Rock breaks scissors",
    "Paper covers rock",
    "Scissors cut paper",
]

say <<"EOT"
\n>> Rock Paper Scissors <<\n
** Enter 'r', 'p', or 's' as your play.
** Enter 'q' to exit the game.
** Running score shown as <your wins>:<my wins>
EOT

var plays   = 0
var aScore  = 0
var pScore  = 0
var pcf     = [0,0,0]      # pcf = player choice frequency
var aChoice = pick(0..2)   # ai choice for first play is completely random

loop {
    var pi = Sys.scanln("Play: ")
    pi == 'q' && break

    var pChoice = rps.index(pi)

    if (pChoice == -1) {
        STDERR.print("Invalid input!\n")
        next
    }

    ++pcf[pChoice]
    ++plays

    # show result of play
    ">> My play: %-8s".printf(rps[aChoice])

    given ((aChoice - pChoice + 3) % 3) {
        when (0) { say "Tie." }
        when (1) { "%-*s %s".printlnf(30, msg[aChoice], 'My point');   aScore++ }
        when (2) { "%-*s %s".printlnf(30, msg[pChoice], 'Your point'); pScore++ }
    }

    # show score
    "%-6s".printf("%d:%d" % (pScore, aScore))

    # compute ai choice for next play
    given (plays.rand.int) { |rn|
        case (rn < pcf[0])        { aChoice = 1 }
        case (pcf[0]+pcf[1] > rn) { aChoice = 2 }
        default                   { aChoice = 0 }
    }
}
```

Sample run:

Output:

```
>> Rock Paper Scissors <<

** Enter 'r', 'p', or 's' as your play.
** Enter 'q' to exit the game.
** Running score shown as <your wins>:<my wins>

Play: r
>> My play: s       Rock breaks scissors           Your point
1:0   Play: p
>> My play: p       Tie.
1:0   Play: r
>> My play: s       Rock breaks scissors           Your point
2:0   Play: s
>> My play: p       Scissors cut paper             Your point
3:0   Play: r
>> My play: p       Paper covers rock              My point
3:1   Play: p
>> My play: r       Paper covers rock              Your point
4:1   Play: q
```

Roman numerals/Decode

```
func roman2arabic(roman) {

    var arabic = 0
    var last_digit = 1000

    static m = Hash(
        I =>    1,
        V =>    5,
        X =>   10,
        L =>   50,
        C =>  100,
        D =>  500,
        M => 1000,
    )

    roman.uc.chars.map{m{_} \\ 0}.each { |digit|
        if (last_digit < digit) {
            arabic -= (2 * last_digit)
        }
        arabic += (last_digit = digit)
    }

    return arabic
}

%w(MCMXC MMVIII MDCLXVI).each { |roman_digit|
    "%-10s == %d\n".printf(roman_digit, roman2arabic(roman_digit))
}
```

Output:

```
MCMXC      == 1990
MMVIII     == 2008
MDCLXVI    == 1666
```

Simpler:

```
func roman2arabic(digit) {
    digit.uc.trans([
        :M:  '1000+',
        :CM:  '900+',
        :D:   '500+',
        :CD:  '400+',
        :C:   '100+',
        :XC:   '90+',
        :L:    '50+',
        :XL:   '40+',
        :X:    '10+',
        :IX:    '9+',
        :V:     '5+',
        :IV:    '4+',
        :I:     '1+',
    ]).split('+').map{.to_i}.sum;
}

%w(MCMXC MMVIII MDCLXVI).each { |roman_num|
    say "#{roman_num}\t-> #{roman2arabic(roman_num)}";
}
```

Output:

```
MCMXC   -> 1990
MMVIII  -> 2008
MDCLXVI -> 1666
```

Roman numerals/Encode

```
func arabic2roman(num, roman='') {
    static lookup = [
        :M:1000, :CM:900, :D:500,
        :CD:400, :C:100,  :XC:90,
        :L:50,   :XL:40,  :X:10,
        :IX:9,   :V:5,    :IV:4,
        :I:1
    ]
    lookup.each { |pair|
        while (num >= pair.second) {
            roman += pair.first
            num -= pair.second
        }
    }
    return roman
}
say("1990 in roman is " + arabic2roman(1990))
say("2008 in roman is " + arabic2roman(2008))
say("1666 in roman is " + arabic2roman(1666))
```

Output:

```
1990 in roman is MCMXC
2008 in roman is MMVIII
1666 in roman is MDCLXVI
```

Roots of a function

```
func f(x) {
    x*x*x - 3*x*x + 2*x
}

var step = 0.001
var start = -1
var stop = 3

for x in range(start+step, stop, step) {
    static sign = false
    given (var value = f(x)) {
        when (0) {
            say "Root found at #{x}"
        }
        case (sign && ((value > 0) != sign)) {
            say "Root found near #{x}"
        }
    }
    sign = value>0
}
```

Output:

```
Root found at 0
Root found at 1
Root found at 2
```

Roots of a quadratic function

```
var sets = [
            [1,    2,  1],
            [1,    2,  3],
            [1,   -2,  1],
            [1,    0, -4],
            [1, -1e6,  1],
           ]

func quadroots(a, b, c) {
    var root = sqrt(b**2 - 4*a*c)

    [(-b + root) / (2 * a),
     (-b - root) / (2 * a)]
}

sets.each { |coefficients|
    say ("Roots for #{coefficients}",
        "=> (#{quadroots(coefficients...).join(', ')})")
}
```

Output:

```
Roots for [1, 2, 1]=> (-1, -1)
Roots for [1, 2, 3]=> (-1+1.414213562373095048801688724209698078569671875381, -1-1.41421356237309504880
Roots for [1, -2, 1]=> (1, 1)
Roots for [1, 0, -4]=> (2, -2)
Roots for [1, -1000000, 1]=> (999999.99999899999999999899999999999979999999999995, 0.000001000000000001000
```

Roots of unity

```
func roots_of_unity(n) {
    n.of { |j|
        exp(2i * Num.pi / n * j)
    }
}

roots_of_unity(5).each { |c|
    printf("%+.5f%+.5fi\n", c.reals)
}
```

Output:

```
+1.00000+0.00000i
+0.30902+0.95106i
-0.80902+0.58779i
-0.80902-0.58779i
+0.30902-0.95106i
```

Rosetta Code/Count examples

```
var lwp = require('LWP::UserAgent').new(agent => 'Mozilla/5.0');

var site = 'http://rosettacode.org';
var list_url = '/mw/api.php?action=query&list=categorymembers&'+
               'cmtitle=Category:Programming_Tasks&cmlimit=500&format=xml';

var content = lwp.get(site + list_url).decoded_content;

while (var m = content.match(/cm.*?title="(.*?)"/g)) {
    (var slug = m[0]).gsub!(' ', '_');
    var count = lwp.get("#{site}/wiki/#{slug}").decoded_content.count(/toclevel-1/g);
    say "#{m[0]}: #{count} examples";
}
```

Output:

```
100 doors: 2180 examples
24 game: 760 examples
24 game/Solve: 450 examples
9 billion names of God the integer: 320 examples
99 Bottles of Beer: 2330 examples
A+B: 1800 examples
ABC Problem: 720 examples
Abstract type: 680 examples
...
```

Rosetta Code/Fix code tags

```
var langs = %w(ada cpp-qt pascal lscript z80 visualprolog
html4strict cil objc asm progress teraterm hq9plus genero tsql
email pic16 tcl apt_sources io apache vhdl avisynth winbatch
vbnet ini scilab ocaml-brief sas actionscript3 qbasic perl bnf
cobol powershell php kixtart visualfoxpro mirc make javascript
cpp sdlbasic cadlisp php-brief rails verilog xml csharp
actionscript nsis bash typoscript freebasic dot applescript
haskell dos oracle8 cfdg glsl lotusscript mpasm latex sql klonec
ruby ocaml smarty python oracle11 caddcl robots groovy smalltalk
diff fortran cfm lua modula3 vb autoit java text scala
lotusformulas pixelbender reg _div whitespace providex asp css
lolcode lisp inno mysql plsql matlab oobas vim delphi xorg_conf
gml prolog bf per scheme mxml d basic4gl m68k gnuplot idl abap
intercal c_mac thinbasic java5 xpp boo klonecpp blitzbasic eiffel
povray c gettext).join('|')

var text = ARGF.slurp

text.gsub!(Regex('<(' + langs + ')>'), {|s1| "<lang #{s1}>" })
text.gsub!(Regex('</(' + langs + ')>'), "</" + "lang>")
text.gsub!(
    Regex('<code\h+(' + langs + ')>(.*?)</code>', 's'),
    {|s1,s2| "<lang #{s1}>#{s2}</" + "lang>"}
)

print text
```

Rosetta Code/Rank languages by popularity

```
require('MediaWiki::API')

var api = %O<MediaWiki::API>.new(
    Hash(api_url => 'http://rosettacode.org/mw/api.php')
)

var languages = []
var gcmcontinue
loop {
    var apih = api.api(
        Hash(
            action      => 'query',
            generator   => 'categorymembers',
            gcmtitle    => 'Category:Programming Languages',
            gcmlimit    => 250,
            prop        => 'categoryinfo',
            gcmcontinue => gcmcontinue,
        )
    )

    languages.append(apih{:query}{:pages}.values...)
    gcmcontinue = apih{:continue}{:gcmcontinue}
    gcmcontinue || break
}

languages.each { |lang|
    lang{:title} -= /^Category:/
    lang{:categoryinfo}{:size} := 0
}

var sorted_languages = languages.sort_by { |lang|
    -lang{:categoryinfo}{:size}
}

sorted_languages.each_kv { |i, lang|
    printf("%3d. %20s - %3d\n", i+1, lang{:title}, lang{:categoryinfo}{:size})
}
```

Output:

```
  1.               Racket - 938
  2.               Python - 930
  3.                  Tcl - 904
  4.               Perl 6 - 877
  5.                    J - 863
  6.                  Zkl - 834
  7.                 Ruby - 830
  8.                    C - 805
  9.                   Go - 793
 10.              Haskell - 785
 11.                 Java - 783
 12.                 REXX - 776
 13.                 Perl - 765
 14.                    D - 753
 15.             PicoLisp - 731
 16.          Mathematica - 702
 17.                Sidef - 696
...
```

Rot-13

```
# Returns a copy of 's' with rot13 encoding.
func rot13(s) {
    s.tr('A-Za-z', 'N-ZA-Mn-za-m')
}

# Perform rot13 on standard input.
STDIN.each { |line| say rot13(line) }
```

RSA code

```
const n = 9516311845790656153499716760847001433441357
const e = 65537
const d = 5617843187844953170308463622230283376298685

module Message {
    var alphabet = [('A' .. 'Z')..., ' ']
    var rad = alphabet.len
    var code = Hash({|i| (alphabet[i], i) }.map(^rad)...)
    func encode(String t) {
        [code{t.reverse.chars...}] ~Z* t.len.range.map { |i| rad**i } -> sum
    }
    func decode(Number n) {
        ''.join(alphabet[
            gather {
                loop {
                    var (d, m) = divmod(n, rad)
                    take(m)
                    break if (n < rad)
                    n = d
                }
            }...]
        ).reverse
    }
}

var secret_message = "ROSETTA CODE"
say "Secret message is #{secret_message}"

var numeric_message = Message::encode(secret_message)
say "Secret message in integer form is #{numeric_message}"

var numeric_cipher = expmod(numeric_message, e, n)
say "After exponentiation with public exponent we get: #{numeric_cipher}"

var text_cipher = Message::decode(numeric_cipher)
say "This turns into the string #{text_cipher}"

var numeric_cipher2 = Message::encode(text_cipher)
say "If we re-encode it in integer form we get #{numeric_cipher2}"

var numeric_message2 = expmod(numeric_cipher2, d, n)
say "After exponentiation with SECRET exponent we get: #{numeric_message2}"

var secret_message2 = Message::decode(numeric_message2)
say "This turns into the string #{secret_message2}"
```

Output:

```
Secret message is ROSETTA CODE
Secret message in integer form is 97525102075211938
After exponentiation with public exponent we get: 832617177411398382204524348895631875839642 6
This turns into the string ZULYDCEZOWTFXFRRNLIMGNUPHVCJSX
If we re-encode it in integer form we get 832617177411398382204524348895631875839642 6
After exponentiation with SECRET exponent we get: 97525102075211938
This turns into the string ROSETTA CODE
```

Run-length encoding

First solution:

```
func encode(str) {
  str.gsub(/((.)(\2*))/, {|a,b| "#{a.len}#{b}" })
}

func decode(str) {
  str.gsub(/(\d+)(.)/, {|a,b| b * a.to_i })
}
```

Output:

```
12W1B12W3B24W1B14W
```

Second solution, encoding the length into a byte:

```
func encode(str) {
    str.gsub(/(.)(\1{0,254})/, {|a,b| b.len+1 -> chr + a })
}

func decode(str) {
     var chars = str.chars
     var r = ''
     {|i|
         r += (chars[2*i + 1] * chars[2*i].ord)
     } << ^(chars.len//2)
     return r
}
```

Output:

```
"\fW\1B\fW\3B\30W\1B\16W"
```

Run as a daemon or service

When the "daemon" argument is specified, a fork of the program is created with its STDOUT redirected into the file "foo.txt", and the main process is exited.

```
var block = {
    for n in (1..100) {
        STDOUT.say(n)
        Sys.sleep(0.5)
    }
}

if (ARGV[0] == 'daemon') {
    STDERR.say("Daemon mode")
    STDOUT{:fh} = %f'foo.txt'.open_w(){:fh}
    STDOUT.autoflush(true)
    block.fork
    STDERR.say("Exiting")
    Sys.exit(0)
}

STDERR.say("Normal mode")
block.run
```

Runge-Kutta method

```
func runge_kutta(yp) {
    func (t, y, δt) {
        var a = (δt * yp(t, y))
        var b = (δt * yp(t + δt/2, y + a/2))
        var c = (δt * yp(t + δt/2, y + b/2))
        var d = (δt * yp(t + δt, y + c));
        (a + 2*(b + c) + d) / 6
    }
}

define δt = 0.1
var δy = runge_kutta(func(t, y) { t * y.sqrt })

var(t, y) = (0, 1)
loop {
    t.is_int &&
        printf("y(%2d) = %12f ± %e\n", t, y, abs(y - ((t**2 + 4)**2 / 16)))
    t <= 10 || break
    y += δy(t, y, δt)
    t += δt
}
```

Output:

```
y( 0) =     1.000000 ± 0.000000e+00
y( 1) =     1.562500 ± 1.457219e-07
y( 2) =     3.999999 ± 9.194792e-07
y( 3) =    10.562497 ± 2.909562e-06
y( 4) =    24.999994 ± 6.234909e-06
y( 5) =    52.562489 ± 1.081970e-05
y( 6) =    99.999983 ± 1.659460e-05
y( 7) =   175.562476 ± 2.351773e-05
y( 8) =   288.999968 ± 3.156520e-05
y( 9) =   451.562459 ± 4.072316e-05
y(10) =   675.999949 ± 5.098329e-05
```

Runtime evaluation

The eval method evaluates a string as code and returns the resulting object.

```
var (a, b) = (-5, 7)
say eval 'abs(a * b)'    # => 35
say abs(a * b)           # => 35
```

Runtime evaluation/In an environment

```
func eval_with_x(code, x, y) {
    var f = eval(code)
    x = y
    eval(code) - f
}

say eval_with_x(x: 3, y: 5, code: '2 ** x')   # => 24
```

Ruth-Aaron numbers

```
say "First 30 Ruth-Aaron numbers (factors):"
say 30.by {|n| (sopfr(n) == sopfr(n+1)) && (n > 0) }.join(' ')

say "\nFirst 30 Ruth-Aaron numbers (divisors):"
say 30.by {|n| ( sopf(n) ==  sopf(n+1)) && (n > 0) }.join(' ')
```

Output:

```
First 30 Ruth-Aaron numbers (factors):
5 8 15 77 125 714 948 1330 1520 1862 2491 3248 4185 4191 5405 5560 5959 6867 8280 8463 10647 12351 1458

First 30 Ruth-Aaron numbers (divisors):
5 24 49 77 104 153 369 492 714 1682 2107 2299 2600 2783 5405 6556 6811 8855 9800 12726 13775 18655 2118
```

S-expressions

```
func sexpr(txt) {
    txt.trim!

    if (txt.match(/^\((.*)\)$/s)) {|m|
        txt = m[0]
    }
    else {
        die "Invalid: <<#{txt}>>"
    }

    var w
    var ret = []

    while (!txt.is_empty) {
        given (txt.first) {
            when('(') {
                (w, txt) = txt.extract_bracketed('()');
                w = sexpr(w)
            }
            when ('"') {
                (w, txt) = txt.extract_delimited('"')
                w.sub!(/^"(.*)"/, {|s1| s1 })
            }
            else {
                txt.sub!(/^(\S+)/, {|s1| w = s1; '' })
            }
        }
        ret << w
        txt.trim_beg!
    }
    return ret
}

func sexpr2txt(String e) {
    e ~~ /[\s"\(\)]/ ? do { e.gsub!('"', '\\"'); %Q("#{e}") } : e
}

func sexpr2txt(expr) {
    '(' + expr.map {|e| sexpr2txt(e) }.join(' ') + ')'
}

var s = sexpr(%q{

((data "quoted data" 123 4.5)
 (data (!@# (4.5) "(more" "data)")))

})

say s               # dump structure
say sexpr2txt(s)    # convert back
```

Output:

```
[["data", "quoted data", "123", "4.5"], ["data", ["!@#", ["4.5"], "(more", "data)"]]]
((data "quoted data" 123 4.5) (data (!@# (4.5) "(more" "data)")))
```

Safe and Sophie Germain primes

```
^Inf -> lazy.grep{|p| all_prime(p, 2*p + 1) }.first(50).slices(10).each{
    .join(', ').say
}
```

Output:

```
2, 3, 5, 11, 23, 29, 41, 53, 83, 89
113, 131, 173, 179, 191, 233, 239, 251, 281, 293
359, 419, 431, 443, 491, 509, 593, 641, 653, 659
683, 719, 743, 761, 809, 911, 953, 1013, 1019, 1031
1049, 1103, 1223, 1229, 1289, 1409, 1439, 1451, 1481, 1499
```

Safe primes and unsafe primes

```
func is_safeprime(p) {
    is_prime(p) && is_prime((p-1)/2)
}

func is_unsafeprime(p) {
    is_prime(p) && !is_prime((p-1)/2)
}

func safeprime_count(from, to) {
    from..to -> count_by(is_safeprime)
}

func unsafeprime_count(from, to) {
    from..to -> count_by(is_unsafeprime)
}

say "First 35 safe-primes:"
say (1..Inf -> lazy.grep(is_safeprime).first(35).join(' '))
say ''
say "First 40 unsafe-primes:"
say (1..Inf -> lazy.grep(is_unsafeprime).first(40).join(' '))
say ''
say "There are #{safeprime_count(1, 1e6)} safe-primes bellow 10^6"
say "There are #{unsafeprime_count(1, 1e6)} unsafe-primes bellow 10^6"
say ''
say "There are #{safeprime_count(1, 1e7)} safe-primes bellow 10^7"
say "There are #{unsafeprime_count(1, 1e7)} unsafe-primes bellow 10^7"
```

Output:

```
First 35 safe-primes:
5 7 11 23 47 59 83 107 167 179 227 263 347 359 383 467 479 503 563 587 719 839 863 887 983 1019 1187 12

First 40 unsafe-primes:
2 3 13 17 19 29 31 37 41 43 53 61 67 71 73 79 89 97 101 103 109 113 127 131 137 139 149 151 157 163 173

There are 4324 safe-primes bellow 10^6
There are 74174 unsafe-primes bellow 10^6

There are 30657 safe-primes bellow 10^7
There are 633922 unsafe-primes bellow 10^7
```

Sailors, coconuts and a monkey problem

```
func coconuts(sailors, monkeys=1) {
    if ((sailors < 2) || (monkeys < 1) || (sailors <= monkeys)) {
        return 0
    }
    var blue_cocos = sailors-1
    var pow_bc = blue_cocos**sailors
    var x_cocos = pow_bc
    while ((x_cocos-blue_cocos)%sailors || ((x_cocos-blue_cocos)/sailors < 1)) {
        x_cocos += pow_bc
    }
    return monkeys*(x_cocos / pow_bc * sailors**sailors - blue_cocos)
}

for sailor in (2..9) {
    say "#{sailor}: #{coconuts(sailor)}";
}
```

Output:

```
2: 11
3: 25
4: 765
5: 3121
6: 233275
7: 823537
8: 117440505
9: 387420481
```

Same fringe

```
var trees = [
    # 0..2 are same
    [ 'd', [ 'c', [ 'a', 'b', ], ], ],
    [ [ 'd', 'c' ], [ 'a', 'b' ] ],
    [ [ [ 'd', 'c', ], 'a', ], 'b', ],
    # and this one's different!
    [ [ [ [ [ 'a' ], 'b' ], 'c', ], 'd', ], 'e', ], 'f' ],
]

func get_tree_iterator(*rtrees) {
    var tree
    func {
        tree = rtrees.pop
        while (defined(tree) && tree.kind_of(Array)) {
            rtrees.append(tree[1])
            tree = tree[0]
        }
        return tree
    }
}

func cmp_fringe(a, b) {
    var ti1 = get_tree_iterator(a)
    var ti2 = get_tree_iterator(b)
    loop {
        var (L, R) = (ti1(), ti2())
         defined(L) &&  defined(R) && (L == R) && next
        !defined(L) && !defined(R) && return "Same"
        return "Different"
    }
}

for idx in ^(trees.end) {
    say ("tree[#{idx}] vs tree[#{idx+1}]: ",
           cmp_fringe(trees[idx], trees[idx+1]))
}
```

Output:

```
tree[0] vs tree[1]: Same
tree[1] vs tree[2]: Same
tree[2] vs tree[3]: Different
```

Sattolo cycle

Modifies the array in-place:

```
func sattolo_cycle(arr) {
    for i in (arr.len ^.. 1) {
        arr.swap(i, pick(^i))
    }
}
```

Scope/Function names and labels

In Sidef, the same rule which is applied to variable scoping, is applied to functions and classes as well, which means that a

function defined inside another function is not visible outside the current scope.

```
# Nested functions
func outer {
    func inner {};   # not visible outside
}

# Nested classes
class Outer {
    class Inner {};  # not visisble outside
}
```

Search a list

```
var haystack = %w(Zig Zag Wally Ronald Bush Krusty Charlie Bush Bozo)

%w(Bush Washington).each { |needle|
    var i = haystack.first_index{|item| item == needle}
    if (i >= 0) {
        say "#{i} #{needle}"
    } else {
        die "#{needle} is not in haystack"
    }
}
```

Output:

```
4 Bush
Washington is not in haystack at find.sf line 9.
```

Extra credit:

```
var haystack = %w(Zig Zag Wally Ronald Bush Krusty Charlie Bush Bozo)
say haystack.last_index{|item| item == "Bush"}
```

Output:

```
7
```

Search a list of records

```
struct City {
    String name,
    Number population,
}

var cities = [
    City("Lagos", 21),
    City("Cairo", 15.2),
    City("Kinshasa-Brazzaville", 11.3),
    City("Greater Johannesburg", 7.55),
    City("Mogadishu", 5.85),
    City("Khartoum-Omdurman", 4.98),
    City("Dar Es Salaam", 4.7),
    City("Alexandria", 4.58),
    City("Abidjan", 4.4),
    City("Casablanca", 3.98),
]

say cities.index{|city| city.name == "Dar Es Salaam"}   # => 6
say cities.first{|city| city.population < 5.0}.name     # => Khartoum-Omdurman
```

Secure temporary file

```
var tmpfile = require('File::Temp')
var fh = tmpfile.new(UNLINK => 0)
say fh.filename
fh.print("Hello, World!\n")
fh.close
```

SEDOLs

```
func sedol(s) {

    die 'No vowels allowed' if (s ~~ /[AEIOU]/)
    die 'Invalid format'    if (s !~ /^[0-9B-DF-HJ-NP-TV-Z]{6}$/)

    const base36 = [[(^10)..., ('A'..'Z')...], ^36].zip.flat.to_h
    const weights = [1, 3, 1, 7, 3, 9]

    var vs = [base36{ s.chars... }]
    var checksum = (vs ~Z* weights -> sum)
    var check_digit = ((10 - checksum%10) % 10)
    return (s + check_digit)
}

%w(
    710889
    B0YBKJ
    406566
    B0YBLH
    228276
    B0YBKL
    557910
    B0YBKR
    585284
    B0YBKT
    B00030
).each { |s|
    say sedol(s)
}
```

Output:

```
7108899
B0YBKJ7
4065663
B0YBLH2
2282765
B0YBKL9
5579107
B0YBKR5
5852842
B0YBKT7
B000300
```

Self-describing numbers

```
func sdn(Number n) {
    var b = [0]*n.len
    var a = n.digits.flip
    a.each { |i| b[i] := 0 ++ }
    a == b
}

var values = [1210, 2020, 21200, 3211000,
42101000, 521001000, 6210001000, 27, 115508]

values.each { |test|
    say "#{test} is #{sdn(test) ? '' : 'NOT ' }a self describing number."
}

say "\nSelf-descriptive numbers less than 1e5 (in base 10):"
{|i| say i if sdn(i) } << ^1e5
```

Output:

```
1210 is a self describing number.
2020 is a self describing number.
21200 is a self describing number.
3211000 is a self describing number.
42101000 is a self describing number.
521001000 is a self describing number.
6210001000 is a self describing number.
27 is NOT a self describing number.
115508 is NOT a self describing number.

Self-descriptive numbers less than 1e5 (in base 10):
1210
2020
21200
```

Extra credit: this will generate all the self-describing numbers in bases 7 to 36:

```
for b in (7..36) {
    var n = ((b-4) * b**(b-1) + 2*(b**(b-2)) + b**(b-3) + b**3 -> base(b))
    say "base #{'%2d' % b}: #{n}"
}
```

Output:

```
base  7: 3211000
base  8: 42101000
base  9: 521001000
base 10: 6210001000
base 11: 72100001000
base 12: 821000001000
base 13: 9210000001000
base 14: a2100000001000
base 15: b21000000001000
base 16: c210000000001000
base 17: d2100000000001000
base 18: e21000000000001000
base 19: f210000000000001000
base 20: g2100000000000001000
base 21: h21000000000000001000
base 22: i210000000000000001000
base 23: j2100000000000000001000
base 24: k21000000000000000001000
base 25: l210000000000000000001000
base 26: m2100000000000000000001000
base 27: n21000000000000000000001000
base 28: o210000000000000000000001000
base 29: p2100000000000000000000001000
base 30: q21000000000000000000000001000
base 31: r210000000000000000000000001000
base 32: s2100000000000000000000000001000
base 33: t21000000000000000000000000001000
base 34: u210000000000000000000000000001000
base 35: v2100000000000000000000000000001000
base 36: w21000000000000000000000000000001000
```

Self numbers

Algorithm by David A. Corneth (OEIS: A003052).

```
func is_self_number(n) {

    if (n < 30) {
        return (((n < 10) && (n.is_odd)) || (n == 20))
    }

    var qd = (1 + n.ilog10)
    var r  = (1 + (n-1)%9)
    var h  = (r + 9*(r%2))/2
    var ld = 10

    while (h + 9*qd >= n%ld) {
        ld *= 10
    }

    var vs = idiv(n, ld).sumdigits
    n %= ld

    0..qd -> none { |i|
        vs + sumdigits(n - h - 9*i) == (h + 9*i)
    }
}

say is_self_number.first(50).join(' ')
```

Output:

```
1 3 5 7 9 20 31 42 53 64 75 86 97 108 110 121 132 143 154 165 176 187 198 209 211 222 233 244 255 266 2
```

Simpler algorithm (by M. F. Hasler):

```
func is_self_number(n) {
    1..min(n>>1, 9*n.len) -> none {|i| sumdigits(n-i) == i } && (n > 0)
}
```

Semiprime

Built-in:

```
say is_semiprime(2**128 + 1)   #=> true
say is_semiprime(2**256 - 1)   #=> false
```

User-defined function, with trial division up to a given bound `B`:

```
func is_semiprime(n, B=1e4) {

    with (n.trial_factor(B)) { |f|
        return false if (f.len > 2)
        return f.all { .is_prime } if (f.len == 2)
    }

    n.factor.len == 2
}

say [2,4,99,100,1679,32768,1234567,9876543,900660121].grep(is_semiprime)
```

Output:

```
[4, 1679, 1234567, 900660121]
```

Semordnilap

```
var c = 0
var seen = Hash()

ARGF.each { |line|
    var r = line.reverse
    ((seen{r} := 0 ++) && (c++ < 5) && say "#{line} #{r}") ->
        || (seen{line} := 0 ++)
}

say c
```

Output:

```
$ sidef semordnilap.sf < unixdict.txt
ca ac
dab bad
diva avid
dna and
drab bard
158
```

Send an unknown method call

```
class Example {
    method foo(x) {
        42 + x
    }
}

var name = 'foo'
var obj = Example()

say obj.(name)(5)           # prints: 47
say obj.method(name)(5)     # =//=
```

Separate the house number from the street name

```
var re = %r[
    ( .*? )
    (?:
        \s+
        (
        | \d+ (?: \- | \/ ) \d+
        | (?! 1940 | 1945) \d+ [ a-z I . / \x20 ]* \d*
        )
    )?
$]x

ARGF.each { |line|
    line.chomp!
    if (var m = line.match(re)) {
        printf("%-25s split as (#{m[0]}, #{m[1]})\n", line)
    }
    else {
        warn "Can't parse: «#{line}»"
    }
}
```

Output:

```
Plataanstraat 5           split as (Plataanstraat, 5)
Straat 12                 split as (Straat, 12)
Straat 12 II              split as (Straat, 12 II)
Dr. J. Straat   12        split as (Dr. J. Straat, 12)
Dr. J. Straat 12 a        split as (Dr. J. Straat, 12 a)
Dr. J. Straat 12-14       split as (Dr. J. Straat, 12-14)
Laan 1940 – 1945 37       split as (Laan 1940 – 1945, 37)
Plein 1940 2              split as (Plein 1940, 2)
1213-laan 11              split as (1213-laan, 11)
16 april 1944 Pad 1       split as (16 april 1944 Pad, 1)
1e Kruisweg 36            split as (1e Kruisweg, 36)
Laan 1940-’45 66          split as (Laan 1940-’45, 66)
Laan ’40-’45              split as (Laan ’40-’45, )
Langeloërduinen 3 46      split as (Langeloërduinen, 3 46)
Marienwaerdt 2e Dreef 2   split as (Marienwaerdt 2e Dreef, 2)
Provincialeweg N205 1     split as (Provincialeweg N205, 1)
Rivium 2e Straat 59.      split as (Rivium 2e Straat, 59.)
Nieuwe gracht 20rd        split as (Nieuwe gracht, 20rd)
Nieuwe gracht 20rd 2      split as (Nieuwe gracht, 20rd 2)
Nieuwe gracht 20zw /2     split as (Nieuwe gracht, 20zw /2)
Nieuwe gracht 20zw/3      split as (Nieuwe gracht, 20zw/3)
Nieuwe gracht 20 zw/4     split as (Nieuwe gracht, 20 zw/4)
Bahnhofstr. 4             split as (Bahnhofstr., 4)
Wertstr. 10               split as (Wertstr., 10)
Lindenhof 1               split as (Lindenhof, 1)
Nordesch 20               split as (Nordesch, 20)
Weilstr. 6                split as (Weilstr., 6)
Harthauer Weg 2           split as (Harthauer Weg, 2)
Mainaustr. 49             split as (Mainaustr., 49)
August-Horch-Str. 3       split as (August-Horch-Str., 3)
Marktplatz 31             split as (Marktplatz, 31)
Schmidener Weg 3          split as (Schmidener Weg, 3)
Karl-Weysser-Str. 6       split as (Karl-Weysser-Str., 6)
```

Sequence: nth number with exactly n divisors

```
func f(n {.is_prime}) {
    n.prime**(n-1)
}

func f(n) {
    n.th { .sigma0 == n }
}

say 20.of { f(_+1) }
```

Output:

```
[1, 3, 25, 14, 14641, 44, 24137569, 70, 1089, 405, 819628286980801, 160, 22563490300366186081, 2752, 98
```

Sequence of non-squares

```
func nonsqr(n) { 0.5 + n.sqrt -> floor + n }
{|i| nonsqr(i) }.map(1..22).join(' ').say

{ |i|
  if (nonsqr(i).is_sqr) {
     die "Found a square in the sequence: #{i}"
  }
} << 1..1e6
```

Sequence of primes by Trial Division

Using the *is_prime()* function from: "Primality by trial division"

```
func prime_seq(amount, callback) {
    var (counter, number) = (0, 0)
    while (counter < amount) {
        if (is_prime(number)) {
            callback(number)
            ++counter
        }
        ++number
    }
}

prime_seq(100, {|p| say p })     # prints the first 100 primes
```

Sequence of primorial primes

```
func primorial_primes(n) {

    var k = 1
    var p = 2
    var P = 2

    var seq = []
    for (var i = 0; i < n; ++k) {

        if (is_prime(P-1) || is_prime(P+1)) {
            seq << k
            ++i
        }

        p.next_prime!
        P *= p
    }

    return seq
}

say primorial_primes(20)
```

Output:

```
[1, 2, 3, 4, 5, 6, 11, 13, 24, 66, 68, 75, 167, 171, 172, 287, 310, 352, 384, 457]
```

Sequence: smallest number greater than previous term with exactly n divisors

```
func n_divisors(n, from=1) {
    from..Inf -> first_by { .sigma0 == n }
}

with (1) { |from|
    say 15.of { from = n_divisors(_+1, from) }
}
```

Output:

```
[1, 2, 4, 6, 16, 18, 64, 66, 100, 112, 1024, 1035, 4096, 4288, 4624]
```

Sequence: smallest number with exactly n divisors

```
func n_divisors(n) {
    1..Inf -> first_by { .sigma0 == n }
}

say 15.of { n_divisors(_+1) }
```

Output:

```
[1, 2, 4, 6, 16, 12, 64, 24, 36, 48, 1024, 60, 4096, 192, 144]
```

Set

```
class MySet(*set) {

    method init {
        var elems = set
        set = Hash()
        elems.each { |e| self += e }
    }

    method +(elem) {
        set{elem} = elem
        self
    }

    method del(elem) {
        set.delete(elem)
    }

    method has(elem) {
        set.has_key(elem)
    }

    method ∪(MySet that) {
```

```
        MySet(set.values..., that.values...)
    }

    method ∩(MySet that) {
        MySet(set.keys.grep{ |k| k ∈ that } \
                    .map { |k| set{k} }...)
    }

    method ∖(MySet that) {
        MySet(set.keys.grep{|k| !(k ∈ that) } \
                    .map {|k| set{k} }...)
    }

    method ^(MySet that) {
        var d = ((self ∖ that) ∪ (that ∖ self))
        MySet(d.values...)
    }

    method count { set.len }

    method ≡(MySet that) {
        (self ∖ that -> count.is_zero) && (that ∖ self -> count.is_zero)
    }

    method values { set.values }

    method ⊆(MySet that) {
        that.set.keys.each { |k|
            k ∈ self || return false
        }
        return true
    }

    method to_s {
        "Set{" + set.values.map{|e| "#{e}"}.sort.join(', ') + "}"
    }
}

class Object {
    method ∈(MySet set) {
        set.has(self)
    }
}
```

Usage example:

```
var x = MySet(1, 2, 3)
5..7 -> each { |i| x += i }

var y = MySet(1, 2, 4, x)

say "set x is: #{x}"
say "set y is: #{y}"

[1,2,3,4,x].each { |elem|
    say ("#{elem} is ", elem ∈ y ? '' : 'not', " in y")
}

var (w, z)
say ("union: ", x ∪ y)
say ("intersect: ", x ∩ y)
say ("z = x ∖ y = ", z = (x ∖ y) )
say ("y is ", x ⊆ y ? "" : "not ", "a subset of x")
say ("z is ", x ⊆ z ? "" : "not ", "a subset of x")
say ("z = (x ∪ y) ∖ (x ∩ y) = ", z = ((x ∪ y) ∖ (x ∩ y)))
say ("w = x ^ y = ", w = (x ^ y))
say ("w is ", w ≡ z ? "" : "not ", "equal to z")
say ("w is ", w ≡ x ? "" : "not ", "equal to x")
```

Output:

```
set x is: Set{1, 2, 3, 5, 6, 7}
set y is: Set{1, 2, 4, Set{1, 2, 3, 5, 6, 7}}
1 is  in y
2 is  in y
3 is not in y
4 is  in y
Set{1, 2, 3, 5, 6, 7} is  in y
union: Set{1, 2, 3, 4, 5, 6, 7, Set{1, 2, 3, 5, 6, 7}}
intersect: Set{1, 2}
z = x ∖ y = Set{3, 5, 6, 7}
y is not a subset of x
z is a subset of x
z = (x ∪ y) ∖ (x ∩ y) = Set{3, 4, 5, 6, 7, Set{1, 2, 3, 5, 6, 7}}
w = x ^ y = Set{3, 4, 5, 6, 7, Set{1, 2, 3, 5, 6, 7}}
w is equal to z
w is not equal to x
```

Set consolidation

```
func consolidate() { [] }
func consolidate(this, *those) {
    gather {
        consolidate(those...).each { |that|
            if (this & that) { this |= that }
            else             { take that }
        }
        take this
    }
}

enum |A="A", B, C, D, _E, F, G, H, I, _J, K|

func format(ss) {
    ss.map{ '(' + .join(' ') + ')' }.join(' ')
}

[
    [[A,B], [C,D]],
    [[A,B], [B,D]],
    [[A,B], [C,D], [D,B]],
    [[H,I,K], [A,B], [C,D], [D,B], [F,G,H]]
].each { |ss|
    say (format(ss), "\n\t==> ", format(consolidate(ss...)))
}
```

Output:

```
(A B) (C D)
    ==> (C D) (A B)
(A B) (B D)
    ==> (A B D)
(A B) (C D) (D B)
    ==> (A B C D)
(H I K) (A B) (C D) (D B) (F G H)
    ==> (A B C D) (H I K F G)
```

Seven-sided dice from five-sided dice

```
func dice5 { pick(1..5) }

func dice7 {
  loop {
    var d7 = ((5*dice5() + dice5() - 6) % 8)
    d7 && return d7
  }
}

var count7 = Hash()

var n = 1e6;
n.times { count7{dice7()} := 0 ++ }
count7.keys.sort.each { |k|
    printf("%s: %5.2f%%\n", k, 100*(count7{k}/n * 7 - 1))
}
```

Output:

```
1: -0.00%
2:  0.02%
3:  0.23%
4:  0.42%
5: -0.23%
6: -0.54%
7:  0.10%
```

Sexy primes

```
var limit  = 1e6+35
var primes = limit.primes

say "Total number of primes <= #{limit.commify} is #{primes.len.commify}."
say "Sexy k-tuple primes <= #{limit.commify}:\n"

(2..5).each {|k|
    var groups = []
    primes.each {|p|
        var group = (1..^k -> map {|j| 6*j + p })
        if (group.all{.is_prime} && (group[-1] <= limit)) {
            groups << [p, group...]
        }
    }

    say "...total number of sexy #{k}-tuple primes = #{groups.len.commify}"
    say "...where last 5 tuples are: #{groups.last(5).map{'('+.join(' ')+')'}.join(' ')}\n"
}

var unsexy_primes = primes.grep {|p| is_prime(p+6) || is_prime(p-6) -> not }
say "...total number of unsexy primes = #{unsexy_primes.len.commify}"
say "...where last 10 unsexy primes are: #{unsexy_primes.last(10)}"
```

Output:

```
Total number of primes <= 1,000,035 is 78,500.
Sexy k-tuple primes <= 1,000,035:

...total number of sexy 2-tuple primes = 16,386
...where last 5 tuples are: (999371 999377) (999431 999437) (999721 999727) (999763 999769) (999953 999

...total number of sexy 3-tuple primes = 2,900
...where last 5 tuples are: (997427 997433 997439) (997541 997547 997553) (998071 998077 998083) (99861

...total number of sexy 4-tuple primes = 325
...where last 5 tuples are: (977351 977357 977363 977369) (983771 983777 983783 983789) (986131 986137

...total number of sexy 5-tuple primes = 1
...where last 5 tuples are: (5 11 17 23 29)

...total number of unsexy primes = 48,627
...where last 10 unsexy primes are: [999853, 999863, 999883, 999907, 999917, 999931, 999961, 999979, 99
```

SHA-1

```
var sha = frequire('Digest::SHA')
say sha.sha1_hex('Rosetta Code')
```

Output:

```
48c98f7e5a6e736d790ab740dfc3f51a61abe2b5
```

SHA-256

```
var sha = frequire('Digest::SHA')
say sha.sha256_hex('Rosetta code')
```

Output:

```
764faf5c61ac315f1497f9dfa542713965b785e5cc2f707d6468d7d1124cdfcf
```

Shell one-liner

```
% sidef -E "say 'hello'"
```

Shoelace formula for polygonal area

```
func area_by_shoelace (*p) {
    var x = p.map{_[0]}
    var y = p.map{_[1]}

    var s = (
        (x ~Z* y.rotate(+1)).sum -
        (x ~Z* y.rotate(-1)).sum
    )

    s.abs / 2
}

say area_by_shoelace([3,4], [5,11], [12,8], [9,5], [5,6])
```

Output:

```
30
```

Short-circuit evaluation

```
func a(bool) { print 'A'; return bool }
func b(bool) { print 'B'; return bool }

# Test-driver
func test() {
    for op in ['&&', '||'] {
        for x,y in [[1,1],[1,0],[0,1],[0,0]] {
            "a(%s) %s b(%s): ".printf(x, op, y)
            eval "a(Bool(x)) #{op} b(Bool(y))"
            print "\n"
        }
    }
}

# Test and display
test()
```

Output:

```
a(1) && b(1): AB
a(1) && b(0): AB
a(0) && b(1): A
a(0) && b(0): A
a(1) || b(1): A
a(1) || b(0): A
a(0) || b(1): AB
a(0) || b(0): AB
```

Shortest common supersequence

Uses the *lcs* function defined here.

```
func scs(u, v) {
    var ls = lcs(u, v).chars
    var pat = Regex('(.*)'+ls.join('(.*)')+'(.*)')
    u.scan!(pat)
    v.scan!(pat)
    var ss = (u.shift + v.shift)
    ls.each { |c| ss += (c + u.shift + v.shift) }
    return ss
}

say scs("abcbdab", "bdcaba")
```

Output:

```
abdcabdab
```

Show the (decimal) value of a number of 1s appended with a 3, then squared

```
0..^8 -> each {|n|
    var k = ((10**(n+1) - 1)/9 + 2)
    say [k, k**2]
}
```

Output:

```
[3, 9]
[13, 169]
[113, 12769]
[1113, 1238769]
[11113, 123498769]
[111113, 12346098769]
[1111113, 1234572098769]
[11111113, 123456832098769]
```

Show the epoch

```
say Time(0).gmtime.ctime
```

Output:

```
Thu Jan  1 00:00:00 1970
```

Sierpinski arrowhead curve

Uses the **LSystem()** class from Hilbert curve.

```
var rules = Hash(
    x => 'yF+xF+y',
    y => 'xF-yF-x',
)

var lsys = LSystem(
    width:  550,
    height: 500,

    xoff: -20,
    yoff: -30,

    len:   4,
    turn: -90,
    angle: 60,
    color: 'dark green',
)

lsys.execute('xF', 7, "sierpiński_arrowhead.png", rules)
```

Output image: Sierpiński arrowhead

Sierpinski carpet

```
var c = ['##']
3.times {
    c = (c.map{|x| x * 3 }             +
         c.map{|x| x + ' '*x.len + x } +
         c.map{|x| x * 3 })
}
say c.join("\n")
```

Output:

```
##########################################################
##  ####  ####  ####  ####  ####  ####  ####  ####  ##
##########################################################
######      ############      ############      ######
##  ##      ##  ####  ##      ##  ####  ##      ##  ##
######      ############      ############      ######
##########################################################
##  ####  ####  ####  ####  ####  ####  ####  ####  ##
##########################################################
##################                  ##################
##  ####  ####  ##                  ##  ####  ####  ##
##################                  ##################
######      ######                  ######      ######
##  ##      ##  ##                  ##  ##      ##  ##
######      ######                  ######      ######
##################                  ##################
##  ####  ####  ##                  ##  ####  ####  ##
##################                  ##################
##########################################################
##  ####  ####  ####  ####  ####  ####  ####  ####  ##
##########################################################
######      ############      ############      ######
##  ##      ##  ####  ##      ##  ####  ##      ##  ##
######      ############      ############      ######
##########################################################
##  ####  ####  ####  ####  ####  ####  ####  ####  ##
##########################################################
```

Sierpinski curve

Uses the **LSystem()** class from Hilbert curve.

```
var rules = Hash(
    x => 'xF+G+xF--F--xF+G+x',
)

var lsys = LSystem(
    width:  550,
    height: 550,

    xoff: -9,
    yoff: -271,

    len:   5,
    angle: 45,
    color: 'dark green',
)

lsys.execute('F--xF--F--xF', 5, "sierpiński_curve.png", rules)
```

Output image: Sierpiński curve

Sierpinski pentagon

Generates a SVG image to STDOUT. Redirect to a file to capture and display it.

```
define order = 5
define sides = 5
define dim   = 500
define scaling_factor = ((3 - 5**0.5) / 2)
var orders = order.of {|i| ((1-scaling_factor) * dim) * scaling_factor**i }

say <<"STOP";
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg height="#{dim*2}" width="#{dim*2}"
    style="fill:blue" transform="translate(#{dim},#{dim}) rotate(-18)"
    version="1.1" xmlns="http://www.w3.org/2000/svg">
STOP

var vertices = sides.of {|i| Complex(0, i * Number.tau / sides).exp }

for i in ^(sides**order) {
   var vector = ([vertices["%#{order}d" % i.base(sides) -> chars]] »*« orders «+»)
   var points = (vertices »*» orders[-1]*(1-scaling_factor) »+» vector »reals()» «%« '%0.3f')
   say ('<polygon points="' + points.join(' ') + '"/>')
}

say '</svg>'
```

Sierpinski square curve

Uses the **LSystem()** class from Hilbert curve.

```
var rules = Hash(
    x => 'xF-F+F-xF+F+xF-F+F-x',
)

var lsys = LSystem(
    width:  510,
    height: 510,

    xoff: -505,
    yoff: -254,

    len:   4,
    angle: 90,
    color: 'dark green',
)

lsys.execute('F+xF+F+xF', 5, "sierpiński_square_curve.png", rules)
```

Output image: Sierpiński square curve

Sierpinski triangle

```
func sierpinski_triangle(n) {
    var triangle = ['*']
    { |i|
        var sp = (' ' * 2**i)
        triangle = (triangle.map {|x| sp + x + sp} +
                    triangle.map {|x| x + ' ' + x})
    } * n
    triangle.join("\n")
}

say sierpinski_triangle(4)
```

Output:

```
               *
              * *
             *   *
            * * * *
           *       *
          * *     * *
         *   *   *   *
        * * * * * * * *
       *               *
      * *             * *
     *   *           *   *
    * * * *         * * * *
   *       *       *       *
  * *     * *     * *     * *
 *   *   *   *   *   *   *   *
* * * * * * * * * * * * * * * *
```

Sierpinski triangle/Graphical

Creates a PNG image.

```
func sierpinski_triangle(n) -> Array {
  var triangle = ['*']
  { |i|
    var sp = (' ' * 2**i)
    triangle = (triangle.map {|x| sp + x + sp} +
                triangle.map {|x| x + ' ' + x})
  } * n
  triangle
}

class Array {
  method to_png(scale=1, bgcolor='white', fgcolor='black') {

    static gd = require('GD::Simple')
    var width = self.max_by{.len}.len
    self.map!{|r| "%-#{width}s" % r}

    var img = gd.new(width * scale, self.len * scale)

    for i in ^self {
      for j in RangeNum(i*scale, i*scale + scale) {
        img.moveTo(0, j)
        for line in (self[i].scan(/(\s+|\S+)/)) {
          img.fgcolor(line.contains(/\S/) ? fgcolor : bgcolor)
          img.line(scale * line.len)
        }
      }
    }
    img.png
  }
}

var triangle = sierpinski_triangle(8)
var raw_png = triangle.to_png(bgcolor:'black', fgcolor:'red')
File('triangle.png').write(raw_png, :raw)
```

Sieve of Eratosthenes

```
func sieve(limit) {
    var sieve_arr = [false, false, (limit-1).of(true)...]
    gather {
        sieve_arr.each_kv { |number, is_prime|
            if (is_prime) {
                take(number)
                for i in (number**2 .. limit `by` number) {
                    sieve_arr[i] = false
                }
            }
        }
    }
}

say sieve(100).join(",")
```

Output:

```
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97
```

Alternative implementation:

```
func sieve(limit) {
    var composite = []
    for n in (2 .. limit.isqrt) {
        for i in (n**2 .. limit `by` n) {
            composite[i] = true
        }
    }
    2..limit -> grep{ !composite[_] }
}

say sieve(100).join(",")
```

Simple windowed application

```
require('Gtk2') -> init

# Window.
var window = %O<Gtk2::Window>.new
window.signal_connect('destroy' => { %O<Gtk2>.main_quit })

# VBox.
var vbox = %O<Gtk2::VBox>.new(0, 0)
window.add(vbox)

# Label.
var label = %O<Gtk2::Label>.new('There have been no clicks yet.')
vbox.add(label)

# Button.
var count = 0
var button = %O<Gtk2::Button>.new(' Click Me ')
vbox.add(button)
button.signal_connect('clicked' => {
    label.set_text(++count)
})

# Show.
window.show_all

# Main loop.
%O<Gtk2>.main
```

Simulated annealing

```
module TravelingSalesman {

    # Eₛ: length(path)
    func Eₛ(distances, path) {
        var total = 0
        [path, path.slice(1)].zip {|ci,cj|
            total += distances[ci-1][cj-1]
        }
        total
    }
```

```
    # T: temperature
    func T(k, kmax, kT) { kT * (1 - k/kmax) }

    # ΔE = Eₛ_new - Eₛ_old > 0
    # Prob. to move if ΔE > 0, → 0 when T → 0 (fronzen state)
    func P(ΔE, k, kmax, kT) { exp(-ΔE / T(k, kmax, kT)) }

    # ΔE from path ( .. a u b .. c v d ..) to (.. a v b ... c u d ..)
    # ΔE before swapping (u,v)
    # Quicker than Eₛ(s_next) - Eₛ(path)
    func dE(distances, path, u, v) {

        var a = distances[path[u-1]-1][path[u]-1]
        var b = distances[path[u+1]-1][path[u]-1]
        var c = distances[path[v-1]-1][path[v]-1]
        var d = distances[path[v+1]-1][path[v]-1]

        var na = distances[path[u-1]-1][path[v]-1]
        var nb = distances[path[u+1]-1][path[v]-1]
        var nc = distances[path[v-1]-1][path[u]-1]
        var nd = distances[path[v+1]-1][path[u]-1]

        if (v == u+1) {
            return ((na+nd) - (a+d))
        }

        if (u == v+1) {
            return ((nc+nb) - (c+b))
        }

        return ((na+nb+nc+nd) - (a+b+c+d))
    }

    const dirs = [1, -1, 10, -10, 9, 11, -11, -9]

    func _prettypath(path) {
        path.slices(10).map { .map{ "%3s" % _ }.join(', ') }.join("\n")
    }

    func findpath(distances, kmax, kT) {

        const n = distances.len
        const R = 2..n

        var path = [1, R.shuffle..., 1]
        var Emin = Eₛ(distances, path)

        printf("# Entropy(s₀) = s%10.2f\n", Emin)
        printf("# Random path:\n%s\n\n", _prettypath(path))

        for k in (1 .. kmax) {

            if (k % (kmax//10) == 0) {
                printf("k: %10d | T: %8.4f | Eₛ: %8.4f\n", k, T(k, kmax, kT), Eₛ(distances, path))
            }

            var u = R.rand
            var v = (path[u-1] + dirs.rand)
            v ~~ R || next

            var δE = dE(distances, path, u-1, v-1)
            if ((δE < 0) || (P(δE, k, kmax, kT) >= 1.rand)) {
                path.swap(u-1, v-1)
                Emin += δE
            }
```

```
        }

        printf("k: %10d | T: %8.4f | Eₛ: %8.4f\n", kmax, T(kmax, kmax, kT), Eₛ(distances, path))
        say ("\n# Found path:\n", _prettypath(path))
        return path
    }
}

var citydist = {|ci|
    { |cj|
        var v1 = Vec(ci%10, ci//10)
        var v2 = Vec(cj%10, cj//10)
        v1.dist(v2)
    }.map(1..100)
}.map(1..100)

TravelingSalesman::findpath(citydist, 1e6, 1)
```

Output:

```
# Entropy(s₀) =     520.29
# Random path:
  1,  10,  79,  52,  24,   9,  58,  11,  42,   4
 15,  87,  62,  88,  21,  91,  99,  84,  61,  14
  5,  17,  33,  95,  74,  31,  40,  13,  37,  69
  6,  22,  97,  45,  56,  63,  75,  83,  53,  41
  3,  47,  89,  80,  78,  98,  46,  18,  25,  51
 93,  16,  50,  30,  48,   8,  66,  68,  59,  73
 49,  96,  36,  32, 100,  27,  76,  44,  64,  39
 90,  82,  20,  12,  54,  86,  29,  81,  26,  72
 60,  94,  35,  92,  43,   7,  85,  55,  28,  57
 23,  34,  65,  71,  38,   2,  77,  70,  19,  67
  1

k:     100000 | T:   0.9000 | Eₛ: 185.1809
k:     200000 | T:   0.8000 | Eₛ: 168.6262
k:     300000 | T:   0.7000 | Eₛ: 146.5948
k:     400000 | T:   0.6000 | Eₛ: 140.1441
k:     500000 | T:   0.5000 | Eₛ: 129.5132
k:     600000 | T:   0.4000 | Eₛ: 132.8942
k:     700000 | T:   0.3000 | Eₛ: 124.2865
k:     800000 | T:   0.2000 | Eₛ: 120.0859
k:     900000 | T:   0.1000 | Eₛ: 115.0771
k:    1000000 | T:   0.0000 | Eₛ: 114.9728
k:    1000000 | T:   0.0000 | Eₛ: 114.9728

# Found path:
  1,   2,  13,   3,   4,   5,   6,   7,   8,   9
 19,  29,  18,  28,  27,  17,  16,  26,  25,  15
 14,  24,  23,  12,  11,  10,  20,  21,  30,  40
 41,  31,  32,  44,  45,  46,  47,  48,  49,  39
 38,  37,  36,  35,  34,  42,  51,  50,  60,  61
 52,  53,  54,  55,  56,  57,  58,  59,  69,  68
 77,  67,  66,  65,  64,  62,  72,  71,  70,  80
 81,  82,  74,  75,  76,  87,  88,  78,  79,  89
 99,  98,  97,  96,  86,  85,  83,  91,  90, 100
 92,  93,  94,  95,  84,  73,  63,  43,  33,  22
  1
```

Singleton

```
class Singleton(name) {
    static instance

    method new(name) {
        instance := Singleton.bless(Hash(:name => name))
    }
    method new {
        Singleton.new(nil)
    }
}

var s1 = Singleton('foo')
say s1.name                 #=> 'foo'
say s1.object_id            #=> '30424504'

var s2 = Singleton()
say s2.name                 #=> 'foo'
say s2.object_id            #=> '30424504'

s2.name = 'bar'             # change name in s2
say s1.name                 #=> 'bar'
```

Singly-linked list/Element definition

```
var node = Hash(
    data => 'say what',
    next => foo_node,
)

node{:next} = bar_node   # mutable
```

Singly-linked list/Element insertion

```
func insert_after(a,b) {
    b{:next} = a{:next}
    a{:next} = b
}

var B = Hash(
    data => 3,
    next => nil,    # not a circular list
)
var A = Hash(
    data => 1,
    next => B,
)
var C = Hash(
    data => 2,
)

insert_after(A, C)
```

Singly-linked list/Traversal

```
var list = 'a':'b':'c':nil
#var list = ['a', ['b', ['c']]]
#var list = Pair('a', Pair('b', Pair('c', nil)))

for (var l = list; l != nil; l = l[1]) {
    say l[0]
}
```

Output:

```
a
b
c
```

Sleep

```
var sec = read(Number)        # any positive number (it may be fractional)
say "Sleeping for #{sec} sec..."
Sys.sleep(sec)                # in seconds
#Sys.usleep(sec)              # in microseconds
#Sys.nanosleep(sec)           # in nanoseconds
say "Awake!"
```

Smallest numbers

```
0..50 -> map {|n| 1..Inf -> first {|k| Str(k**k).contains(n) } }.say
```

Output:

```
[9, 1, 3, 5, 2, 4, 4, 3, 7, 9, 10, 11, 5, 19, 22, 26, 8, 17, 16, 19, 9, 8, 13, 7, 17, 4, 17, 3, 11, 18,
```

Smallest square that begins with n

```
1..49 -> map {|n|
    [n, n.isqrt..Inf -> first {|j| Str(j**2).starts_with(n) }]
}.slices(5).each {|a|
    say a.map { '%2d: %5d %-8s' % (_[0], _[1]**2, "(#{_[1]}^2)") }.join(' ')
}
```

Output:

```
 1:     1 (1^2)     2:    25 (5^2)     3:    36 (6^2)     4:     4 (2^2)     5:   529 (23^2)
 6:    64 (8^2)     7:   729 (27^2)    8:    81 (9^2)     9:     9 (3^2)    10:   100 (10^2)
11:  1156 (34^2)   12:   121 (11^2)   13:  1369 (37^2)   14:   144 (12^2)   15:  1521 (39^2)
16:    16 (4^2)    17:  1764 (42^2)   18:  1849 (43^2)   19:   196 (14^2)   20:  2025 (45^2)
21:  2116 (46^2)   22:   225 (15^2)   23:  2304 (48^2)   24:  2401 (49^2)   25:    25 (5^2)
26:  2601 (51^2)   27:  2704 (52^2)   28:   289 (17^2)   29:  2916 (54^2)   30:  3025 (55^2)
31:  3136 (56^2)   32:   324 (18^2)   33:  3364 (58^2)   34:  3481 (59^2)   35: 35344 (188^2)
36:    36 (6^2)    37:  3721 (61^2)   38:  3844 (62^2)   39:  3969 (63^2)   40:   400 (20^2)
41: 41209 (203^2)  42:  4225 (65^2)   43:  4356 (66^2)   44:   441 (21^2)   45: 45369 (213^2)
46:  4624 (68^2)   47:  4761 (69^2)   48:   484 (22^2)   49:    49 (7^2)
```

Smarandache prime-digital sequence

```
func is_prime_digital(n) {
    n.is_prime && n.digits.all { .is_prime }
}

say is_prime_digital.first(25).join(',')
say is_prime_digital.nth(100)
```

Output:

```
2,3,5,7,23,37,53,73,223,227,233,257,277,337,353,373,523,557,577,727,733,757,773,2237,2273
33223
```

Smith numbers

```
var primes = Enumerator({ |callback|
    static primes = Hash()
    var p = 2
    loop {
        callback(p)
        p = (primes{p} := p.next_prime)
    }
})

func factors(remainder) {

    remainder == 1 && return([remainder])

    gather {
        primes.each { |factor|
            if (factor*factor > remainder) {
                take(remainder) if (remainder > 1)
                break
            }

            while (factor.divides(remainder)) {
                take(factor)
                break if ((remainder /= factor) == 1)
            }
        }
    }
}

func is_smith_number(n) {
    !n.is_prime && (n.sumdigits == factors(n).join.to_i.sumdigits)
}

var s = range(2, 10_000).grep { is_smith_number(_) }
say "#{s.len} Smith numbers below 10_000"
say "First 10: #{s.first(10)}"
say "Last  10: #{s.last(10)}"
```

Output:

```
376 Smith numbers below 10_000
First 10: [4, 22, 27, 58, 85, 94, 121, 166, 202, 265]
Last  10: [9843, 9849, 9861, 9880, 9895, 9924, 9942, 9968, 9975, 9985]
```

Snake

```
class SnakeGame(w, h) {
    const readkey = frequire('Term::ReadKey')
    const ansi    = frequire('Term::ANSIColor')

    enum (VOID, HEAD, BODY, TAIL, FOOD)

    define (
        LEFT  = [+0, -1],
        RIGHT = [+0, +1],
        UP    = [-1, +0],
        DOWN  = [+1, +0],
    )

    define BG_COLOR    = "on_black"
```

```
define FOOD_COLOR  = ("red"        + " " + BG_COLOR)
define SNAKE_COLOR = ("bold green" + " " + BG_COLOR)
define SLEEP_SEC   = 0.02

const (
    A_VOID  = ansi.colored(' ', BG_COLOR),
    A_FOOD  = ansi.colored('*', FOOD_COLOR),
    A_BLOCK = ansi.colored('■', SNAKE_COLOR),
)

has dir = LEFT
has grid = [[]]
has head_pos = [0, 0]
has tail_pos = [0, 0]

method init {
    grid = h.of { w.of { [VOID] } }

    head_pos = [h//2, w//2]
    tail_pos = [head_pos[0], head_pos[1]+1]

    grid[head_pos[0]][head_pos[1]] = [HEAD, dir]    # head
    grid[tail_pos[0]][tail_pos[1]] = [TAIL, dir]    # tail

    self.make_food()
}

method make_food {
    var (food_x, food_y)

    do {
        food_x = w.rand.int
        food_y = h.rand.int
    } while (grid[food_y][food_x][0] != VOID)

    grid[food_y][food_x][0] = FOOD
}

method display {
    print("\033[H", grid.map { |row|
        row.map { |cell|
            given (cell[0]) {
                when (VOID) { A_VOID }
                when (FOOD) { A_FOOD }
                default     { A_BLOCK }
            }
          }.join('')
        }.join("\n")
    )
}

method move {
    var grew = false

    # Move the head
    var (y, x) = head_pos...

    var new_y = (y+dir[0] % h)
    var new_x = (x+dir[1] % w)

    var cell = grid[new_y][new_x]

    given (cell[0]) {
        when (BODY) { die "\nYou just bit your own body!\n" }
        when (TAIL) { die "\nYou just bit your own tail!\n" }
        when (FOOD) { grew = true; self.make_food()         }
```

```
        }

        # Create a new head
        grid[new_y][new_x] = [HEAD, dir]

        # Replace the current head with body
        grid[y][x] = [BODY, dir]

        # Update the head position
        head_pos = [new_y, new_x]

        # Move the tail
        if (!grew) {
            var (y, x) = tail_pos...

            var pos   = grid[y][x][1]
            var new_y = (y+pos[0] % h)
            var new_x = (x+pos[1] % w)

            grid[y][x][0]         = VOID    # erase the current tail
            grid[new_y][new_x][0] = TAIL    # create a new tail

            tail_pos = [new_y, new_x]
        }
    }

    method play {
        STDOUT.autoflush(true)
        readkey.ReadMode(3)

        try {
            loop {
                var key
                while (!defined(key = readkey.ReadLine(-1))) {
                    self.move()
                    self.display()
                    Sys.sleep(SLEEP_SEC)
                }

                given (key) {
                    when ("\e[A") { if (dir != DOWN ) { dir = UP    } }
                    when ("\e[B") { if (dir != UP   ) { dir = DOWN  } }
                    when ("\e[C") { if (dir != LEFT ) { dir = RIGHT } }
                    when ("\e[D") { if (dir != RIGHT) { dir = LEFT  } }
                }
            }
        }
        catch {
            readkey.ReadMode(0)
        }
    }
}

var w = `tput cols`.to_i
var h = `tput lines`.to_i

SnakeGame(w || 80, h || 24).play
```

Sockets

```
var host = Socket.gethostbyname('localhost')
var in = Socket.sockaddr_in(256, host)
var proto = Socket.getprotobyname('tcp')

var sock = Socket.open(Socket.AF_INET, Socket.SOCK_STREAM, proto)
sock.connect(in)
sock.send('hello socket world', 0, in)
sock.close
```

Solve triangle solitare puzzle

```
const N = [0,1,1,1,1,1,1,1,1,1,1,1,1,1]

const G = [
    [ 0, 1, 3],[ 0, 2, 5],[ 1, 3, 6],
    [ 1, 4, 8],[ 2, 4, 7],[ 2, 5, 9],
    [ 3, 4, 5],[ 3, 6,10],[ 3, 7,12],
    [ 4, 7,11],[ 4, 8,13],[ 5, 8,12],
    [ 5, 9,14],[ 6, 7, 8],[ 7, 8, 9],
    [10,11,12],[11,12,13],[12,13,14],
]

const format = ({"#{' '*(5-_)}#{'%d '*_}\n"}.map(1..5).join + "\n")

func solve(n, i, g) is cached {
    i == N.end && return "Solved"
    n[g[1]] == 0 && return nil

    var s = given(n[g[0]]) {
        when(0) {
            n[g[2]] == 0 && return nil
            "#{g[2]} to #{g[0]}\n"
        }
        default {
            n[g[2]] == 1 && return nil
            "#{g[0]} to #{g[2]}\n"
        }
    }

    var a = n.clone
    g.each {|n| a[n] = 1-a[n] }
    var r = ''
    G.each {|g| (r = solve(a, i+1, g)) && break }
    r ? (s + (format % (a...)) + r) : r
}

format.printf(N...)

var r = ''
G.each {|g| (r = solve(N, 1, g)) && break }
say (r ? r : "No solution found")
```

Sort a list of object identifiers

```
func sort_OIDs(ids) {
    ids.sort_by { |id|
        id.split('.').map { Num(_) }
    }
}

var OIDs = %w(
    1.3.6.1.4.1.11.2.17.19.3.4.0.10
    1.3.6.1.4.1.11.2.17.5.2.0.79
    1.3.6.1.4.1.11.2.17.19.3.4.0.4
    1.3.6.1.4.1.11150.3.4.0.1
    1.3.6.1.4.1.11.2.17.19.3.4.0.1
    1.3.6.1.4.1.11150.3.4.0
)

sort_OIDs(OIDs).each { .say }
```

Output:

```
1.3.6.1.4.1.11.2.17.5.2.0.79
1.3.6.1.4.1.11.2.17.19.3.4.0.1
1.3.6.1.4.1.11.2.17.19.3.4.0.4
1.3.6.1.4.1.11.2.17.19.3.4.0.10
1.3.6.1.4.1.11150.3.4.0
1.3.6.1.4.1.11150.3.4.0.1
```

Sort an array of composite structures

```
# Declare an array of pairs
var people = [['joe', 120], ['foo', 31], ['bar', 51]]

# Sort the array in-place by name
people.sort! {|a,b| a[0] <=> b[0] }

# Alternatively, we can use the `.sort_by{}` method
var sorted = people.sort_by { |item| item[0] }

# Display the sorted array
say people
```

Output:

```
[["bar", 51], ["foo", 31], ["joe", 120]]
```

Sort an integer array

```
var nums = [2,4,3,1,2]
var sorted = nums.sort   # returns a new sorted array.
nums.sort!               # sort 'nums' in-place (at the variable level)
```

Sort disjoint sublist

```
func disjointSort(values, indices) {
    values[indices.sort] = [values[indices]].sort...
}

var values =  [7, 6, 5, 4, 3, 2, 1, 0]
var indices = [6, 1, 7]

disjointSort(values, indices)
say values
```

Output:

```
[7, 0, 5, 4, 3, 2, 1, 6]
```

Sort numbers lexicographically

```
func lex_order (n) {
    [range(1, n, n.sgn)...].sort_by { Str(_) }
}

[13, 21, -22].each {|n|
    printf("%4s: %s\n", n, lex_order(n))
}
```

Output:

```
  13: [1, 10, 11, 12, 13, 2, 3, 4, 5, 6, 7, 8, 9]
  21: [1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2, 20, 21, 3, 4, 5, 6, 7, 8, 9]
-22: [-1, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -2, -20, -21, -22, -3, -4, -5, -6, -7, -8,
```

Sort primes from list to a list

```
var arr = [2,43,81,122,63,13,7,95,103]
say arr.grep{.is_prime}.sort
```

Output:

```
[2, 7, 13, 43, 103]
```

Sort stability

Sidef uses the stable merge-sort algorithm for sorting an array.

```
var table = [
  <UK  London>,
  <US  New\ York>,
  <US  Birmingham>,
  <UK  Birmingham>,
]

table.sort {|a,b| a[0] <=> b[0] }.each { |col|
    say "#{col[0]} #{col[1]}"
}
```

Output:

```
UK London
UK Birmingham
US New York
US Birmingham
```

Sort three variables

Generalized solution, for an arbitrary number of variable references:

```
func sort_refs(*arr) {
    arr.map{ *_ }.sort ~Z arr -> each { *_[1] = _[0] }
}

var x = 77444
var y =   -12
var z =     0

sort_refs(\x, \y, \z)

say x
say y
say z
```

Output:

```
-12
0
77444
```

Alternatively, without using a sorting function:

```
var x = 77444
var y =   -12
var z =     0

(x, y) = (y, x) if (x > y)
(x, z) = (z, x) if (x > z)
(y, z) = (z, y) if (y > z)

say x
say y
say z
```

Output:

```
-12
0
77444
```

Sort using a custom comparator

```
func mycmp(a, b) { (b.len <=> a.len) || (a.lc <=> b.lc) }
var strings = %w(Here are some sample strings to be sorted)
var sorted = strings.sort(mycmp)
```

Sorting algorithms/Bead sort

```
func beadsort(arr) {

    var rows = []
    var columns = []

    for datum in arr {
        for column in ^datum {
            ++(columns[column] := 0)
            ++(rows[columns[column] - 1] := 0)
        }
    }

    rows.reverse
}

say beadsort([5,3,1,7,4,1,1])
```

Output:

```
[1, 1, 1, 3, 4, 5, 7]
```

Sorting algorithms/Bogosort

```
func in_order(a) {
    return true if (a.len <= 1)
    var first = a[0]
    a.ft(1).all { |elem| first <= elem  ? do { first = elem; true } : false }
}

func bogosort(a) {
    a.shuffle! while !in_order(a)
    return a
}

var arr = 5.of{ 100.irand }
say "Before: #{arr}"
say "After:  #{bogosort(arr)}"
```

Output:

```
Before: 57 45 83 85 33
After:  33 45 57 83 85
```

Sorting algorithms/Bubble sort

```
func bubble_sort(arr) {
    loop {
        var swapped = false
        { |i|
            if (arr[i] > arr[i+1]) {
                arr[i, i+1] = arr[i+1, i]
                swapped = true
            }
        } << ^arr.end
        swapped || break
    }
    return arr
}
```

Sorting Algorithms/Circle Sort

```
func circlesort(arr, beg=0, end=arr.end) {
    var swaps = 0
    if (beg < end) {
        var (lo, hi) = (beg, end)
        do {
            if (arr[lo] > arr[hi]) {
                arr.swap(lo, hi)
                ++swaps
            }
            ++hi if (--hi == ++lo)
        } while (lo < hi)
        swaps += circlesort(arr, beg, hi)
        swaps += circlesort(arr, lo, end)
    }
    return swaps
}

var numbers = %n(2 3 3 5 5 1 1 7 7 6 6 4 4 0 0)
do { say numbers } while circlesort(numbers)

var strs = ["John", "Kate", "Zerg", "Alice", "Joe", "Jane", "Alice"]
do { say strs } while circlesort(strs)
```

Output:

```
[2, 3, 3, 5, 5, 1, 1, 7, 7, 6, 6, 4, 4, 0, 0]
[0, 0, 1, 4, 1, 5, 3, 7, 2, 3, 4, 5, 6, 6, 7]
[0, 0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 7, 6, 6, 7]
[0, 0, 1, 1, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7]
["John", "Kate", "Zerg", "Alice", "Joe", "Jane", "Alice"]
["Alice", "Jane", "Alice", "Joe", "John", "Kate", "Zerg"]
["Alice", "Alice", "Jane", "Joe", "John", "Kate", "Zerg"]
```

Sorting algorithms/Cocktail sort

```
func cocktailsort(a) {
    var swapped = false
    func cmpsw(i) {
        if (a[i] > a[i+1]) {
            a[i, i+1] = a[i+1, i]
            swapped = true
        }
    }
    var max = a.end
    do {
        {|i| cmpsw(i) } << ^max
        swapped.not! && break
        {|i| cmpsw(max-i) } << 1..max
    } while (swapped)
    return a
}
```

Test:

```
var numbers = [7,6,5,9,8,4,3,1,2,0]
say cocktailsort(numbers)

var strs = ["John", "Kate", "Zerg", "Alice", "Joe", "Jane"]
say cocktailsort(strs)
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
['Alice', 'Jane', 'Joe', 'John', 'Kate', 'Zerg']
```

Sorting algorithms/Comb sort

```
func comb_sort(arr) {
    var gap = arr.len
    var swaps = true
    while (gap > 1 || swaps) {
        gap.div!(1.25).int! if (gap > 1)
        swaps = false
        for i in ^(arr.len - gap) {
            if (arr[i] > arr[i+gap]) {
                arr[i, i+gap] = arr[i+gap, i]
                swaps = true
            }
        }
    }
    return arr
}
```

Sorting algorithms/Counting sort

```
func counting_sort(a, min, max) {
    var cnt = ([0] * (max - min + 1))
    a.each {|i| cnt[i-min]++ }
    cnt.map {|i| [min++] * i }.flat
}

var a = 100.of { 100.irand }
say counting_sort(a, 0, 100)
```

Sorting algorithms/Cycle sort

```
func cycle_sort (array) {
    var (writes=0, pos=0)

    func f(i, Ref item, bool=false) {
        pos = (i + array.ft(i+1).count{ _ < *item })
        return(false) if (bool && pos==i)
        while (*item == array[pos]) { ++pos }
        (array[pos], *item) = (*item, array[pos])
        ++writes
        return true
    }

    array.each_kv { |i, item|
        f(i, \item, true) || next
        while (pos != i) {
            f(i, \item)
        }
    }

    return writes
}

var a = %n(0 1 2 2 2 2 1 9 3.5 5 8 4 7 0 6)

say a.join(' ')
say ('writes ', cycle_sort(a))
say a.join(' ')
```

Output:

```
0 1 2 2 2 2 1 9 3.5 5 8 4 7 0 6
writes 10
0 0 1 1 2 2 2 2 3.5 4 5 6 7 8 9
```

Sorting algorithms/Gnome sort

```
class Array {
    method gnomesort {
        var (i=1, j=2)
        var len = self.len
        while (i < len) {
            if (self[i-1] <= self[i]) {
                (i, j) = (j, j+1)
            }
            else {
                self[i-1, i] = self[i, i-1]
                if (--i == 0) {
                    (i, j) = (j, j+1)
                }
            }
        }
        return self
    }
}

var ary = [7,6,5,9,8,4,3,1,2,0]
say ary.gnomesort
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Sorting algorithms/Heapsort

```
func sift_down(a, start, end) {
    var root = start
    while ((2*root + 1) <= end) {
        var child = (2*root + 1)
        if ((child+1 <= end) && (a[child] < a[child + 1])) {
            child += 1
        }
        if (a[root] < a[child]) {
            a[child, root] = a[root, child]
            root = child
        } else {
            return nil
        }
    }
}

func heapify(a, count) {
    var start = ((count - 2) / 2)
    while (start >= 0) {
        sift_down(a, start, count-1)
        start -= 1
    }
}

func heap_sort(a, count) {
    heapify(a, count)
    var end = (count - 1)
    while (end > 0) {
        a[0, end] = a[end, 0]
        end -= 1
        sift_down(a, 0, end)
    }
    return a
}

var arr = (1..10 -> shuffle)   # creates a shuffled array
say arr                        # prints the unsorted array
heap_sort(arr, arr.len)        # sorts the array in-place
say arr                        # prints the sorted array
```

Output:

```
[10, 5, 2, 1, 7, 6, 4, 8, 3, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Sorting algorithms/Insertion sort

```
class Array {
    method insertion_sort {
        { |i|
            var j = i-1
            var k = self[i]
            while ((j >= 0) && (k < self[j])) {
                self[j+1] = self[j]
                j--
            }
            self[j+1] = k
        } << 1..self.end
        return self
    }
}

var a = 10.of { 100.irand }
say a.insertion_sort
```

Sorting algorithms/Merge sort

```
func merge(left, right) {
    var result = []
    while (left && right) {
        result << [right,left].min_by{.first}.shift
    }
    result + left + right
}

func mergesort(array) {
    var len = array.len
    len < 2 && return array

    var (left, right) = array.part(len//2)

    left  = __FUNC__(left)
    right = __FUNC__(right)

    merge(left, right)
}

# Numeric sort
var nums = rand(1..100, 10)
say mergesort(nums)

# String sort
var strings = rand('a'..'z', 10)
say mergesort(strings)
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7]
['a', 'b', 'c', 'd', 'e']
```

Sorting algorithms/Pancake sort

```
func pancake(a) {
    for idx in ^(a.end) {
        var min = idx
        for i in (idx+1 .. a.end) { min = i if (a[min] > a[i]) }
        next if (a[min] == a[idx])
        a[min..a.end] = [a[min..a.end]].reverse...
        a[idx..a.end] = [a[idx..a.end]].reverse...
    }
    return a
}

var arr = 10.of{ 100.irand }
say "Before: #{arr}"
say "After:  #{pancake(arr)}"
```

Output:

```
Before: 61 29 68 15 34 2 32 54 73 43
After:  2 15 29 32 34 43 54 61 68 73
```

Sorting algorithms/Patience sort

```
func patience(deck) {
  var stacks = [];
  deck.each { |card|
    given (stacks.first { card < .last }) { |stack|
      case (defined stack) {
        stack << card
      }
      default {
        stacks << [card]
      }
    }
  }

  gather {
    while (stacks) {
      take stacks.min_by { .last }.pop
      stacks.grep!{ !.is_empty }
    }
  }
}

var a = [4, 65, 2, -31, 0, 99, 83, 782, 1]
say patience(a)
```

Output:

```
[-31, 0, 1, 2, 4, 65, 83, 99, 782]
```

Sorting algorithms/Permutation sort

```
func psort(x, d=x.end) {

    if (d == 0) {
        for i in (1 .. x.end) {
            (x[i] < x[i-1]) && return false
        }
        return true
    }

    (d+1).times {
        x.prepend(x.splice(d, 1)...)
        x[d] < x[d-1] && next
        psort(x, d-1) && return true
    }

    return false
}

var a = 10.of { 100.irand }
say "Before:\t#{a}"
psort(a)
say "After:\t#{a}"
```

Output:

```
Before: 60 98 85 85 37 0 62 96 95 2
After:  0 2 37 60 62 85 85 95 96 98
```

Sorting algorithms/Quicksort

```
func quicksort (a) {
    a.len < 2 && return(a)
    var p = a.pop_rand           # to avoid the worst cases
    __FUNC__(a.grep{ .< p}) + [p] + __FUNC__(a.grep{ .>= p})
}

say quicksort(rand(1..100, 10))
```

Sorting algorithms/Radix sort

```
class Array {
    method radix_sort(base=10) {
        var arr = self.clone
        var rounds = ([arr.minmax].map{.abs}.max.ilog(base) + 1)
        for i in (0..rounds) {
            var buckets = (2*base -> of {[]})
            var base_i = base**i
            for n in arr {
                var digit = (n/base_i % base)
                digit += base if (0 <= n)
                buckets[digit].append(n)
            }
            arr = buckets.flat
        }
        return arr
    }
}

for arr in [
    [1, 3, 8, 9, 0, 0, 8, 7, 1, 6],
    [170, 45, 75, 90, 2, 24, 802, 66],
    [170, 45, 75, 90, 2, 24, -802, -66],
    [100000, -10000, 400, 23, 10000],
] {
    say arr.radix_sort
}
```

Output:

```
[0, 0, 1, 1, 3, 6, 7, 8, 8, 9]
[2, 24, 45, 66, 75, 90, 170, 802]
[-802, -66, 2, 24, 45, 75, 90, 170]
[-10000, 23, 400, 10000, 100000]
```

Sorting algorithms/Selection sort

```
class Array {
    method selectionsort {
        for i in ^(self.end) {
            var min_idx = i
            for j in (i+1 .. self.end) {
                if (self[j] < self[min_idx]) {
                    min_idx = j
                }
            }
            self.swap(i, min_idx)
        }
        return self
    }
}

var nums = [7,6,5,9,8,4,3,1,2,0]
say nums.selectionsort

var strs = ["John", "Kate", "Zerg", "Alice", "Joe", "Jane"]
say strs.selectionsort
```

Sorting algorithms/Shell sort

```
func shell_sort(a) {
    var h = a.len
    while (h >>= 1) {
        for i in (h .. a.end) {
            var k = a[i]
            for (var j = i; (j >= h) && (k < a[j - h]); j -= h) {
                a[j] = a[j - h]
            }
            a[j] = k
        }
    }
    return a
}

var a = rand(1..100, 10)
say a
shell_sort(a)
say a
```

Output:

```
[54, 67, 65, 8, 56, 83, 64, 42, 20, 17]
[8, 17, 20, 42, 54, 56, 64, 65, 67, 83]
```

Sorting algorithms/Sleep sort

```
ARGV.map{.to_i}.map{ |i|
    { Sys.sleep(i); say i }.fork
}.each{.wait}
```

Output:

```
% sidef test.sf 5 1 3 2 11 6 4
1
2
3
4
5
6
11
```

Sorting algorithms/Stooge sort

```
func stooge(x, i, j) {
    if (x[j] < x[i]) {
        x.swap(i, j)
    }

    if (j-i > 1) {
        var t = ((j - i + 1) / 3)
        stooge(x, i,     j - t)
        stooge(x, i + t, j    )
        stooge(x, i,     j - t)
    }
}

var a = 10.of { 100.irand }

say "Before #{a}"
stooge(a, 0, a.end)
say "After  #{a}"
```

Sorting algorithms/Strand sort

```
func merge(x, y) {
    var out = []
    while (x && y) {
        given (x[-1] <=> y[-1]) {
            when ( 1) { out.prepend(x.pop) }
            when (-1) { out.prepend(y.pop) }
            default   { out.prepend(x.pop, y.pop) }
        }
    }
    x + y + out
}

func strand(x) {
    x || return []
    var out = [x.shift]
    if (x.len) {
        for i in (-x.len .. -1) {
            if (x[i] >= out[-1]) {
                out.append(x.pop_at(i))
            }
        }
    }
    return out
}

func strand_sort(x) {
    var out = []
    while (var strd = strand(x)) {
        out = merge(out, strd)
    }
    return out
}

var a = 10.of { 100.irand }
say "Before: #{a}"
say "After: #{strand_sort(a)}"
```

Output:

```
Before: 24 62 29 95 11 21 46 3 23 20
After: 3 11 20 21 23 24 29 46 62 95
```

Soundex

```
func soundex(word, length=3) {

    # Uppercase the argument passed in to normalize it
    # and drop any non-alphabetic characters
    word.uc!.tr!('A-Z', '', 'cd')

    # Return if word does not contain 'A-Z'
    return(nil) if (word.is_empty)

    var firstLetter = word.char(0)

    # Replace letters with corresponding number values
    word.tr!('BFPV',     '1', 's')
    word.tr!('CGJKQSXZ', '2', 's')
    word.tr!('DT',       '3', 's')
    word.tr!('L',        '4', 's')
    word.tr!('MN',       '5', 's')
    word.tr!('R',        '6', 's')

    # Discard the first letter
    word.ft!(1)

    # Remove A, E, H, I, O, U, W, and Y
    word.tr!('AEHIOUWY', '', 'd')

    # Return the soundex code
    firstLetter + (word.chars + length.of('0') -> first(length).join)
}

func testSoundex {

    # Key-value pairs of names and corresponding Soundex codes
    var sndx = Hash(
                "Euler"               => "E4600",
                "Gauss"               => "G2000",
                "Hilbert"             => "H4163",
                "Knuth"               => "K5300",
                "Lloyd"               => "L3000",
                "Lukasieicz"          => "L2220",
                'fulkerson'           => 'F4262',
                'faulkersuhn'         => 'F4262',
                'fpfffffauhlkkersssin' => 'F4262',
                'Aaeh'                => 'A0000',
               )

    sndx.keys.sort.each { |name|
        var findSdx = soundex(name, 4)
        say "The soundex for #{name} should be #{sndx{name}} and is #{findSdx}"
        if (findSdx != sndx{name}) {
            say "\tHowever, that is incorrect!\n"
        }
    }
}

testSoundex()
```

Sparkline in unicode

```
var bar = @('▁'..'█')
loop {
    print 'Numbers, please, separated by space/commas: '
    var numbers = read(String).trim.split(/[\s,]+/).map{.to_n}
    var (min, max) = numbers.minmax
    say "min: %5f; max: %5f"%(min, max)
    var div = ((max - min) / bar.end)
    say (min == max ? bar.last*numbers.len : numbers.map{|num| bar[(num - min) / div]}.join)
}
```

Output:

```
Numbers, please, separated by space/commas: 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
min: 1.000000; max: 8.000000
▁▂▃▄▅▆▇▇▆▅▄▃▂▁
Numbers, please, separated by space/commas: 1.5, 0.5 3.5, 2.5 5.5, 4.5 7.5, 6.5
min: 0.500000; max: 7.500000
▂▁▃▅▆▇
```

Special divisors

```
1..200 -> grep {|n| n.divisors.all {|d| d.flip `divides` n.flip } }.say
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 17, 19, 22, 23, 26, 27, 29, 31, 33, 37, 39, 41, 43, 44, 46, 47, 53,
```

Special factorials

```
func sf(n) { 1..n -> prod {|k| k! } }
func H(n)  { 1..n -> prod {|k| k**k } }
func af(n) { 1..n -> sum  {|k| (-1)**(n-k) * k! } }
func ef(n) { 1..n -> reduce({|a,b| b**a }, 1) }

func factorial_valuation(n,p) {
    (n - n.sumdigits(p)) / (p-1)
}

func p_adic_inverse (p, k) {

    var n = (k * (p - 1))

    while (factorial_valuation(n, p) < k) {
        n -= (n % p)
        n += p
    }

    return n
}

func rf(f) {

    return nil if (f < 0)
    return 0   if (f <= 1)

    var t = valuation(f, 2) || return nil
    var n = p_adic_inverse(2, t)
    var d = factor(n + 1)[-1]

    if (f.valuation(d) == factorial_valuation(n+1, d)) {
        ++n
    }

    for p in (primes(2, n)) {
        var v = factorial_valuation(n, p)
        f.valuation(p) == v || return nil
        f /= p**v
    }

    (f == 1) ? n : nil
}

say ('sf : ', 10.of(sf).join(' '))
say ('H  : ', 10.of(H).join(' '))
say ('af : ', 10.of(af).join(' '))
say ('ef : ', 5.of(ef).join(' '))

say "ef(5) has #{ef(5).len} digits"

say ('rf : ', [1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800].map(rf))
say ('rf(119) = ', rf(119) \\ 'nil')
say ('rf is defined for: ', 8.by { defined(rf(_)) }.join(', ' ) + ', ...')
```

Output:

```
sf : 1 1 2 12 288 34560 24883200 125411328000 5056584744960000 1834933472251084800000
H  : 1 1 4 108 27648 86400000 4031078400000 3319766398771200000 5569643794172655697920000 215779412229
af : 0 1 1 5 19 101 619 4421 35899 326981
ef : 1 1 2 9 262144
ef(5) has 183231 digits
rf : [0, 2, 3, 4, 5, 6, 7, 8, 9, 10]
rf(119) = nil
rf is defined for: 0, 1, 2, 6, 24, 120, 720, 5040, ...
```

Special neighbor primes

```
func special_neighbor_primes(upto) {
    var list = []
    upto.primes.each_cons(2, {|p1,p2|
        var n = (p1 + p2 - 1)
        if (n.is_prime) {
            list << [p1, p2, n]
        }
    })
    return list
}

with (100) {|n|
    var list = special_neighbor_primes(n)
    say "Found #{list.len} special neighbour primes < n:"
    list.each_2d {|p1,p2,q|
        printf(" (%2s, %2s) => %s\n", p1, p2, q)
    }
}

say ''

for n in (1..7) {
    var list = special_neighbor_primes(10**n)
    say "Found #{list.len} special neighbour primes < 10^#{n}"
}
```

Output:

```
Found 13 special neighbour primes < n:
 ( 3,  5) => 7
 ( 5,  7) => 11
 ( 7, 11) => 17
 (11, 13) => 23
 (13, 17) => 29
 (19, 23) => 41
 (29, 31) => 59
 (31, 37) => 67
 (41, 43) => 83
 (43, 47) => 89
 (61, 67) => 127
 (67, 71) => 137
 (73, 79) => 151

Found 2 special neighbour primes < 10^1
Found 13 special neighbour primes < 10^2
Found 71 special neighbour primes < 10^3
Found 367 special neighbour primes < 10^4
Found 2165 special neighbour primes < 10^5
Found 14526 special neighbour primes < 10^6
Found 103611 special neighbour primes < 10^7
```

Speech synthesis

```
func text2speech(text, lang='en') {
    Sys.run("espeak -v #{lang} -w /dev/stdout #{text.escape} | aplay")
}
text2speech("This is an example of speech synthesis.")
```

Spelling of ordinal numbers

```
var lingua_en = frequire('Lingua::EN::Numbers')
var tests = [1,2,3,4,5,11,65,100,101,272,23456,8007006005004003]

tests.each {|n|
    printf("%16s : %s\n", n, lingua_en.num2en_ordinal(n))
}
```

Output:

```
               1 : first
               2 : second
               3 : third
               4 : fourth
               5 : fifth
              11 : eleventh
              65 : sixty-fifth
             100 : one hundredth
             101 : one hundred and first
             272 : two hundred and seventy-second
           23456 : twenty-three thousand four hundred and fifty-sixth
8007006005004003 : eight quadrillion, seven trillion, six billion, five million, four thousand and thir
```

Spiral matrix

```
func spiral(n) {
    var (x, y, dx, dy, a) = (0, 0, 1, 0, [])
    { |i|
        a[y][x] = i
        var (nx, ny) = (x+dx, y+dy)
        (  if (dx ==  1 && (nx == n || a[ny][nx]!=nil)) { [ 0,  1] }
        elsif (dy ==  1 && (ny == n || a[ny][nx]!=nil)) { [-1,  0] }
        elsif (dx == -1 && (nx  < 0 || a[ny][nx]!=nil)) { [ 0, -1] }
        elsif (dy == -1 && (ny  < 0 || a[ny][nx]!=nil)) { [ 1,  0] }
        else                                            { [dx, dy] }
        ) » (\dx, \dy)
        x = x+dx
        y = y+dy
    } << (1 .. n**2)
    return a
}

spiral(5).each { |row|
    row.map {"%3d" % _}.join(' ').say
}
```

Output:

```
  1   2   3   4   5
 16  17  18  19   6
 15  24  25  20   7
 14  23  22  21   8
 13  12  11  10   9
```

Split a character string based on change of character

```
func group(str) {
    gather {
        while (var match = (str =~ /((.)\g{-1}*)/g)) {
            take(match[0])
        }
    }
}

say group(ARGV[0] \\ 'gHHH5YY++///\\').join(', ')
```

Output:

```
g, HHH, 5, YY, ++, ///, \
```

SQL-based authentication

```
require('DBI')

 # returns a database handle configured to throw an exception on query errors
func connect_db(dbname, host, user, pass) {
    var db = %O<DBI>.connect("dbi:mysql:#{dbname}:#{host}", user, pass)
    db || die (global DBI::errstr)
    db{:RaiseError} = 1
    db
}

 # if the user was successfully created, returns its user id.
 # if the name was already in use, returns nil.
func create_user(db, user, pass) {
    var salt = "C*".pack(16.of { 256.irand }...)
    db.do(
        "INSERT IGNORE INTO users (username, pass_salt, pass_md5)
         VALUES (?, ?, unhex(md5(concat(pass_salt, ?))))", nil, user, salt, pass
    ) ? db{:mysql_insertid} : nil
}

 # if the user is authentic, returns its user id.  otherwise returns nil.
func authenticate_user(db, user, pass) {
    db.selectrow_array("SELECT userid FROM users WHERE
        username=? AND pass_md5=unhex(md5(concat(pass_salt, ?)))",
        nil, user, pass
    )
}
```

Square-free integers

In Sidef, the functions *is_square_free(n)* and *square_free_count(min, max)* are built-in. However, we can very easily reimplement them in Sidef code, as fast integer factorization methods are also available in the language.

```
func is_square_free(n) {

    n.abs!       if (n <  0)
    return false if (n == 0)

    n.factor_exp + [[1,1]] -> all { .[1] == 1 }
}

func square_free_count(n) {
    1 .. n.isqrt -> sum {|k|
        moebius(k) * idiv(n, k*k)
    }
}

func display_results(a, c, f = { _ }) {
    a.each_slice(c, {|*s|
        say s.map(f).join(' ')
    })
}

var a = range(   1,      145).grep {|n| is_square_free(n) }
var b = range(1e12, 1e12+145).grep {|n| is_square_free(n) }

say "There are #{a.len} square-free numbers between 1 and 145:"
display_results(a, 17, {|n| "%3s" % n })

say "\nThere are #{b.len} square-free numbers between 10^12 and 10^12 + 145:"
display_results(b, 5)
say ''

for (2 .. 6) { |n|
    var c = square_free_count(10**n)
    say "The number of square-free numbers between 1 and 10^#{n} (inclusive) is: #{c}"
}
```

Output:

```
There are 90 square-free numbers between 1 and 145:
  1   2   3   5   6   7  10  11  13  14  15  17  19  21  22  23  26
 29  30  31  33  34  35  37  38  39  41  42  43  46  47  51  53  55
 57  58  59  61  62  65  66  67  69  70  71  73  74  77  78  79  82
 83  85  86  87  89  91  93  94  95  97 101 102 103 105 106 107 109
110 111 113 114 115 118 119 122 123 127 129 130 131 133 134 137 138
139 141 142 143 145

There are 89 square-free numbers between 10^12 and 10^12 + 145:
1000000000001 1000000000002 1000000000003 1000000000005 1000000000006
1000000000007 1000000000009 1000000000011 1000000000013 1000000000014
1000000000015 1000000000018 1000000000019 1000000000021 1000000000022
1000000000023 1000000000027 1000000000029 1000000000030 1000000000031
1000000000033 1000000000037 1000000000038 1000000000039 1000000000041
1000000000042 1000000000043 1000000000045 1000000000046 1000000000047
1000000000049 1000000000051 1000000000054 1000000000055 1000000000057
1000000000058 1000000000059 1000000000061 1000000000063 1000000000065
1000000000066 1000000000067 1000000000069 1000000000070 1000000000073
1000000000074 1000000000077 1000000000078 1000000000079 1000000000081
1000000000082 1000000000085 1000000000086 1000000000087 1000000000090
1000000000091 1000000000093 1000000000094 1000000000095 1000000000097
1000000000099 1000000000101 1000000000102 1000000000103 1000000000105
1000000000106 1000000000109 1000000000111 1000000000113 1000000000114
1000000000115 1000000000117 1000000000118 1000000000119 1000000000121
1000000000122 1000000000123 1000000000126 1000000000127 1000000000129
1000000000130 1000000000133 1000000000135 1000000000137 1000000000138
1000000000139 1000000000141 1000000000142 1000000000145

The number of square-free numbers between 1 and 10^2 (inclusive) is: 61
The number of square-free numbers between 1 and 10^3 (inclusive) is: 608
The number of square-free numbers between 1 and 10^4 (inclusive) is: 6083
The number of square-free numbers between 1 and 10^5 (inclusive) is: 60794
The number of square-free numbers between 1 and 10^6 (inclusive) is: 607926
```

Square but not cube

```
var square_and_cube = Enumerator({|f|
    1..Inf -> each {|n| f(n**6) }
})

var square_but_not_cube = Enumerator({|f|
    1..Inf -> lazy.map {|n| n**2 }.grep {|n| !n.is_power(3) }.each {|n| f(n) }
})

say "First 30 positive integers that are a square but not a cube:"
say square_but_not_cube.first(30).join(' ')

say "First 15 positive integers that are both a square and a cube:"
say square_and_cube.first(15).join(' ')
```

Output:

```
First 30 positive integers that are a square but not a cube:
4 9 16 25 36 49 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529 576 625 676 784 841 900 961
First 15 positive integers that are both a square and a cube:
1 64 729 4096 15625 46656 117649 262144 531441 1000000 1771561 2985984 4826809 7529536 11390625
```

Stable marriage problem

```
var he_likes = Hash(
    abe  => < abi eve cath ivy jan dee fay bea hope gay >,
    bob  => < cath hope abi dee eve fay bea jan ivy gay >,
    col  => < hope eve abi dee bea fay ivy gay cath jan >,
    dan  => < ivy fay dee gay hope eve jan bea cath abi >,
    ed   => < jan dee bea cath fay eve abi ivy hope gay >,
    fred => < bea abi dee gay eve ivy cath jan hope fay >,
    gav  => < gay eve ivy bea cath abi dee hope jan fay >,
    hal  => < abi eve hope fay ivy cath jan bea gay dee >,
    ian  => < hope cath dee gay bea abi fay ivy jan eve >,
    jon  => < abi fay jan gay eve bea dee cath ivy hope >,
);

var she_likes = Hash(
    abi  => < bob fred jon gav ian abe dan ed col hal >,
    bea  => < bob abe col fred gav dan ian ed jon hal >,
    cath => < fred bob ed gav hal col ian abe dan jon >,
    dee  => < fred jon col abe ian hal gav dan bob ed >,
    eve  => < jon hal fred dan abe gav col ed ian bob >,
    fay  => < bob abe ed ian jon dan fred gav col hal >,
    gay  => < jon gav hal fred bob abe col ed dan ian >,
    hope => < gav jon bob abe ian dan hal ed col fred >,
    ivy  => < ian col hal gav fred bob abe ed jon dan >,
    jan  => < ed hal gav abe bob jon col ian fred dan >,
);

var guys = he_likes.keys;
var gals = she_likes.keys;

var (:fiancé, :fiancée, :proposed);

func she_prefers (her, hottie) { var a = she_likes{her}; a.index(hottie) < a.index(fiancé{her}) }
func he_prefers  (him, hottie) { var a =  he_likes{him}; a.index(hottie) < a.index(fiancée{him}) }

func unmatched_guy { guys.first {|k| !defined fiancée{k} } }
func preferred_choice(guy) { he_likes{guy}.first {|k| !defined proposed{guy}{k} } }

func engage(guy, gal) {
    fiancé{gal} = guy;
    fiancée{guy} = gal;
}

func match_em {
    say 'Matchmaking:';
    while (defined(var guy = unmatched_guy())) {
        var gal = preferred_choice(guy);
        proposed{guy}{gal} = '❤';
        if (!defined fiancé{gal}) {
            engage(guy, gal);
            say "\t#{gal} and #{guy}";
        }
        elsif (she_prefers(gal, guy)) {
            var engaged_guy = fiancé{gal};
            engage(guy, gal);
            fiancée{engaged_guy} = nil;
            say "\t#{gal} dumped #{engaged_guy} for #{guy}";
        }
    }
}

func check_stability {
```

```
    var instabilities = gather {
        guys.each { |m|
            gals.each { |w|
                if (he_prefers(m, w) && she_prefers(w, m)) {
                    take "\t#{w} prefers #{m} to #{fiancé{w}} and #{m} prefers #{w} to #{fiancée{m}}";
                }
            }
        }
    }

    say 'Stablility:';
    instabilities.is_empty
        ? say "\t(all marriages stable)"
        : instabilities.each { |i| say i };
}

func perturb_em {
    say 'Perturb:';
    say "\tengage abi with fred and bea with jon";
    engage('fred', 'abi');
    engage('jon', 'bea');
}

match_em();
check_stability();

perturb_em();
check_stability();
```

Output:

```
Matchmaking:
        abi and jon
        eve and abe
        bea and fred
        cath and bob
        hope and ian
        eve dumped abe for hal
        ivy and abe
        dee and col
        jan and ed
        fay and dan
        gay and gav
Stablility:
        (all marriages stable)
Perturb:
        engage abi with fred and bea with jon
Stablility:
        gay prefers jon to gav and jon prefers gay to bea
        eve prefers jon to hal and jon prefers eve to bea
        fay prefers jon to dan and jon prefers fay to bea
        bea prefers fred to jon and fred prefers bea to abi
```

Stack

Using a built-in array:

```
var stack = []
stack.push(42)          # pushing
say stack.pop           # popping
say stack.is_empty      # is_emtpy?
```

Creating a Stack class:

```
class Stack(stack=[]) {
    method pop        { stack.pop }
    method push(item) { stack.push(item) }
    method empty      { stack.is_empty }
}

var stack = Stack()
stack.push(42)
say stack.pop           # => 42
say stack.empty         # => true
```

Stair-climbing puzzle

```
func step_up() {
    while (!step()) {
        step_up()
    }
}
```

Standard deviation

Using an object to keep state:

```
class StdDevAccumulator(n=0, sum=0, sumofsquares=0) {
  method <<(num) {
    n += 1
    sum += num
    sumofsquares += num**2
    self
  }

  method stddev {
    sqrt(sumofsquares/n - pow(sum/n, 2))
  }

  method to_s {
    self.stddev.to_s
  }
}

var i = 0
var sd = StdDevAccumulator()
[2,4,4,4,5,5,7,9].each {|n|
    say "adding #{n}: stddev of #{i+=1} samples is #{sd << n}"
}
```

Output:

```
adding 2: stddev of 1 samples is 0
adding 4: stddev of 2 samples is 1
adding 4: stddev of 3 samples is 0.942809041582063365867792482806465385714 3
adding 4: stddev of 4 samples is 0.866025403784438646763723170752936183471 4
adding 5: stddev of 5 samples is 0.979795897113271239278913629882356556786 4
adding 5: stddev of 6 samples is 1
adding 7: stddev of 7 samples is 1.399708424447530341827019471260509366836
adding 9: stddev of 8 samples is 2
```

Using *static* variables:

```
func stddev(x) {
    static(num=0, sum=0, sum2=0)
    num++
    sqrt(
        (sum2 += x**2) / num -
        (((sum += x) / num)**2)
    )
}

%n(2 4 4 4 5 5 7 9).each { say stddev(_) }
```

Output:

```
0
1
0.942809041582063365867792482806465385714 3
0.866025403784438646763723170752936183471 4
0.979795897113271239278913629882356556786 4
1
1.399708424447530341827019471260509366836
2
```

Statistics/Basic

```
func generate_statistics(n) {
    var(sum=0, sum2=0)
    var hist = 10.of(0)

    n.times {
        var r = 1.rand
        sum += r
        sum2 += r**2
        hist[10*r] += 1
    }

    var mean = sum/n
    var stddev = sqrt(sum2/n - mean**2)

    say "size: #{n}"
    say "mean:   #{mean}"
    say "stddev: #{stddev}"

    var max = hist.max
    for i in ^hist {
        printf("%.1f:%s\n", 0.1*i, "=" * 70*hist[i]/max)
    }
    print "\n"
}

[100, 1000, 10000].each {|n| generate_statistics(n) }
```

Statistics/Normal distribution

```
define τ = Num.tau

func normdist (m, σ) {
    var r = sqrt(-2 * 1.rand.log)
    var 0 = (τ * 1.rand)
    r * 0.cos * σ + m
}

var size = 100_000
var mean = 50
var stddev = 4

var dataset = size.of { normdist(mean, stddev) }
var m = (dataset.sum / size)
say ("m: #{m}")

var σ = sqrt(dataset »**» 2 -> sum / size - m**2)
say ("s: #{σ}")

var hash = Hash()
dataset.each { |n| hash{ n.round } := 0 ++ }

var scale = (180 * stddev / size)
const subbar = < | ▏▎▍▌▋▊▉ >

for i in (hash.keys.map{.to_i}.sort) {
    var x = (hash{i} * scale)
    var full = x.int
    var part = (8 * (x - full))
    say (i, "\t", '█' * full, subbar[part])
}
```

Output:

```
m: 50.00446486561320827193555252361187450757867817776
s: 3.98637441101167732229937662764717350787129665319
32 |
33 |
34 |
35 |
36 |
37 |
38 |
39 |
40 |
41 |
42 |
43 |
44 |
45 |
46 |
47 |
48 |
49 |
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
```

Stem-and-leaf plot

```
var data = %i(
     12 127  28  42  39 113  42  18  44 118  44
     37 113 124  37  48 127  36  29  31 125 139
    131 115 105 132 104 123  35 113 122  42 117
    119  58 109  23 105  63  27  44 105  99  41
    128 121 116 125  32  61  37 127  29 113 121
     58 114 126  53 114  96  25 109   7  31 141
     46  13  27  43 117 116  27   7  68  40  31
    115 124  42 128  52  71 118 117  38  27 106
     33 117 116 111  40 119  47 105  57 122 109
    124 115  43 120  43  27  27  18  28  48 125
    107 114  34 133  45 120  30 127  31 116 146
).sort

var stem_unit = 10
var h = data.group_by {|i| i // stem_unit }

var rng = RangeNum(h.keys.map{.to_i}.minmax)
var stem_format =  "%#{rng.min.len.max(rng.max.len)}d"

rng.each { |stem|
    var leafs = (h{stem} \\ [])
    say(stem_format % stem, ' | ', leafs.map { _ % stem_unit }.join(' '))
}
```

Output:

```
 0 | 7 7
 1 | 2 3 8 8
 2 | 3 5 7 7 7 7 7 7 8 8 9 9
 3 | 0 1 1 1 1 2 3 4 5 6 7 7 7 8 9
 4 | 0 0 1 2 2 2 2 3 3 3 4 4 4 5 6 7 8 8
 5 | 2 3 7 8 8
 6 | 1 3 8
 7 | 1
 8 |
 9 | 6 9
10 | 4 5 5 5 5 6 7 9 9 9
11 | 1 3 3 3 3 4 4 4 5 5 5 6 6 6 6 7 7 7 7 8 8 9 9
12 | 0 0 1 1 2 2 3 4 4 4 5 5 5 6 7 7 7 7 8 8
13 | 1 2 3 9
14 | 1 6
```

Stern-Brocot sequence

```
# Declare a function to generate the Stern-Brocot sequence
func stern_brocot {
    var list = [1, 1]
    {
        list.append(list[0]+list[1], list[1])
        list.shift
    }
}

# Show the first fifteen members of the sequence.
say 15.of(stern_brocot()).join(' ')

# Show the (1-based) index of where the numbers 1-to-10 first appears
# in the sequence, and where the number 100 first appears in the sequence.
for i (1..10, 100) {
    var index = 1
    var generator = stern_brocot()
    while (generator() != i) {
        ++index
    }
    say "First occurrence of #{i} is at index #{index}"
}

# Check that the greatest common divisor of all the two consecutive
# members of the series up to the 1000th member, is always one.
var generator = stern_brocot()
var (a, b) = (generator(), generator())
{
    assert_eq(gcd(a, b), 1)
    a = b
    b = generator()
} * 1000

say "All GCD's are 1"
```

Output:

```
1 1 2 1 3 2 3 1 4 3 5 2 5 3 4
First occurrence of 1 is at index 1
First occurrence of 2 is at index 3
First occurrence of 3 is at index 5
First occurrence of 4 is at index 9
First occurrence of 5 is at index 11
First occurrence of 6 is at index 33
First occurrence of 7 is at index 19
First occurrence of 8 is at index 21
First occurrence of 9 is at index 35
First occurrence of 10 is at index 39
First occurrence of 100 is at index 1179
All GCD's are 1
```

Stirling numbers of the first kind

```
func S1(n, k) {     # unsigned Stirling numbers of the first kind
    stirling(n, k).abs
}

const r = (0..12)

var triangle = r.map {|n| 0..n -> map {|k| S1(n, k) } }
var widths   = r.map {|n| r.map {|k| (triangle[k][n] \\ 0).len }.max }

say ('n\k ', r.map {|n| "%*s" % (widths[n], n) }.join(' '))

r.each {|n|
    var str = ('%-3s ' % n)
    str += triangle[n].map_kv {|k,v| "%*s" % (widths[k], v) }.join(' ')
    say str
}

with (100) {|n|
    say "\nMaximum value from the S1(#{n}, *) row:"
    say { S1(n, _) }.map(^n).max
}
```

Output:

```
n\k 0        1         2         3         4        5        6       7      8     9   10 11 12
0   1
1   0        1
2   0        1         1
3   0        2         3         1
4   0        6        11         6         1
5   0       24        50        35        10        1
6   0      120       274       225        85       15        1
7   0      720      1764      1624       735      175       21       1
8   0     5040     13068     13132      6769     1960      322      28      1
9   0    40320    109584    118124     67284    22449     4536     546     36     1
10  0   362880   1026576   1172700    723680   269325    63273    9450    870    45    1
11  0  3628800  10628640  12753576   8409500  3416930   902055  157773  18150  1320   55  1
12  0 39916800 120543840 150917976 105258076 45995730 13339535 2637558 357423 32670 1925 66 1

Maximum value from the S1(100, *) row:
19710908747055261109287881673376044669240511161402863823515728791076863288440277983854056472903481625 29
```

Alternatively, the **S1(n,k)** function can be defined as:

```
func S1((0), (0)) { 1 }
func S1(_, (0))   { 0 }
func S1((0), _)   { 0 }
func S1(n, k) is cached { S1(n-1, k-1) + (n-1)*S1(n-1, k) }
```

Stirling numbers of the second kind

```
func S2(n, k) {     # Stirling numbers of the second kind
    stirling2(n, k)
}

const r = (0..12)

var triangle = r.map {|n| 0..n -> map {|k| S2(n, k) } }
var widths   = r.map {|n| r.map {|k| (triangle[k][n] \\ 0).len }.max }

say ('n\k ', r.map {|n| "%*s" % (widths[n], n) }.join(' '))

r.each {|n|
    var str = ('%-3s ' % n)
    str += triangle[n].map_kv {|k,v| "%*s" % (widths[k], v) }.join(' ')
    say str
}

with (100) {|n|
    say "\nMaximum value from the S2(#{n}, *) row:"
    say { S2(n, _) }.map(^n).max
}
```

Output:

```
n\k 0 1    2     3      4       5       6      7      8     9   10 11 12
0   1
1   0 1
2   0 1    1
3   0 1    3     1
4   0 1    7     6      1
5   0 1   15    25     10       1
6   0 1   31    90     65      15       1
7   0 1   63   301    350     140      21      1
8   0 1  127   966   1701    1050     266     28      1
9   0 1  255  3025   7770    6951    2646    462     36     1
10  0 1  511  9330  34105   42525   22827   5880    750    45    1
11  0 1 1023 28501 145750  246730  179487  63987  11880  1155   55  1
12  0 1 2047 86526 611501 1379400 1323652 627396 159027 22275 1705 66 1

Maximum value from the S2(100, *) row:
76697300535987451552128066127875847873978781283701158409749925701023860862898058480250748224048435451 78
```

Alternatively, the **S2(n,k)** function can be defined as:

```
func S2((0), (0)) { 1 }
func S2(_, (0))   { 0 }
func S2((0), _)   { 0 }
func S2(n, k) is cached { S2(n-1, k)*k + S2(n-1, k-1) }
```

Strange numbers

```
func generate_from_prefix(limit, p, base) {

    var seq = [p]

    for d in (base-1 -> primes) {
        for k in ([1, -1]) {
            var r = p[0]+(k*d)
            next if (r < 0)
            next if (r >= base)
            var t = [r, p...]
            if (t.digits2num(base) <= limit) {
                seq << __FUNC__(limit, t, base)...
            }
        }
    }

    return seq
}

func strange_numbers(limit, base = 10, digits = @(^base)) {
    digits.map {|p| generate_from_prefix(limit, [p], base)... }\
          .map {|t| t.digits2num(base) }\
          .sort.uniq
}

strange_numbers(500).grep { _ > 100 }.slices(10).each{
    .join(' ').say
}
```

Output:

```
130 131 135 136 138 141 142 146 147 149
161 163 164 168 169 181 183 185 186 202
203 205 207 241 242 246 247 249 250 252
253 257 258 270 272 274 275 279 292 294
296 297 302 303 305 307 313 314 316 318
350 352 353 357 358 361 363 364 368 369
381 383 385 386 413 414 416 418 420 424
425 427 429 461 463 464 468 469 470 472
474 475 479 492 494 496 497
```

Strange plus numbers

```
100..500 -> map { .digits }.grep {|d|
    is_prime(d[-1]+d[-2]) && is_prime(d[-2]+d[-3])
}.map{ .digits2num }.slices(10).each { .join(' ').say }
```

Output:

```
111 112 114 116 120 121 123 125 129 141
143 147 149 161 165 167 202 203 205 207
211 212 214 216 230 232 234 238 250 252
256 258 292 294 298 302 303 305 307 320
321 323 325 329 341 343 347 349 383 385
389 411 412 414 416 430 432 434 438 470
474 476 492 494 498
```

Strange unique prime triplets

```
for n in (30, 1000) {
    var triplets = []
    combinations(n.primes, 3, {|*a|
        triplets << a if a.sum.is_prime
    })
    if (n == 30) {
        say "Unique prime triplets (p,q,r) <= #{n} such that p+q+r is prime:"
        triplets.slices(6).each{.join(' ').say}
    }
    printf("Found %d strange unique prime triplets up to %s.\n", triplets.len, n)
}
```

Output:

```
Unique prime triplets (p,q,r) <= 30 such that p+q+r is prime:
[3, 5, 11] [3, 5, 23] [3, 5, 29] [3, 7, 13] [3, 7, 19] [3, 11, 17]
[3, 11, 23] [3, 11, 29] [3, 17, 23] [5, 7, 11] [5, 7, 17] [5, 7, 19]
[5, 7, 29] [5, 11, 13] [5, 13, 19] [5, 13, 23] [5, 13, 29] [5, 17, 19]
[5, 19, 23] [5, 19, 29] [7, 11, 13] [7, 11, 19] [7, 11, 23] [7, 11, 29]
[7, 13, 17] [7, 13, 23] [7, 17, 19] [7, 17, 23] [7, 17, 29] [7, 23, 29]
[11, 13, 17] [11, 13, 19] [11, 13, 23] [11, 13, 29] [11, 17, 19] [11, 19, 23]
[11, 19, 29] [13, 17, 23] [13, 17, 29] [13, 19, 29] [17, 19, 23] [19, 23, 29]
Found 42 strange unique prime triplets up to 30.
Found 241580 strange unique prime triplets up to 1000.
```

Stream merge

```
func merge_streams(streams) {
    var s = streams.map { |stream|
        Pair(stream, stream.readline)
    }.grep {|p| defined(p.value) }

    gather {
        while (s) {
            var p = s.min_by { .value }
            take(p.value)
            p.value = (p.key.readline \\ s.delete_if { _ == p })
        }
    }
}

say merge_streams(ARGV.map {|f| File(f).open_r }).join("\n")
```

String append

```
var str = 'Foo'
str += 'bar'
say str
```

Output:

```
Foobar
```

String case

```
say "alphaBETA".lc              #=> alphabeta
say "alphaBETA".uc              #=> ALPHABETA
say "alphaBETA".tc              #=> AlphaBETA
say "alpha BETA".wc             #=> Alpha Beta
say "alpha BETA".tc             #=> Alpha BETA
say "alpha BETA".tclc           #=> Alpha beta
```

String comparison

```
var methods = %w(== != > >= < <= <=>)
for s1, s2 in [<YUP YUP>,<YUP Yup>,<bot bat>,<aaa zz>] {
    methods.each{|m| "%s %s %s\t%s\n".printf(s1, m, s2, s1.(m)(s2))}
    print "\n"
}
```

String concatenation

```
var s = 'hello'
say s+' literal'
var s1 = s+' literal'
say s1
```

An example of destructive concatenation:

```
s += ' literal'
say s
```

String interpolation (included)

```
var extra = 'little'
say "Mary had a #{extra} lamb"
```

or:

```
say ("Mary had a %s lamb" % 'little')
```

See: documentation

String length

```
var str = "J\x{332}o\x{332}s\x{332}e\x{301}\x{332}"
```

UTF-8 byte length (default):

```
say str.bytes.len       #=> 14
```

UTF-16 byte length:

```
say str.encode('UTF-16').bytes.len      #=> 20
```

```
say str.chars.len    #=> 9
```

```
say str.graphs.len   #=> 4
```

String matching

```
var first = "abc-abcdef-abcd"
var second = "abc"

say first.begins_with(second)       #=> true
say first.contains(second)          #=> true
say first.ends_with(second)         #=> false

# Get and print the location of the match
say first.index(second)             #=> 0

# Find multiple occurrences of a string
var pos = -1
while (pos = first.index(second, pos+1) != -1) {
    say "Match at pos: #{pos}"
}
```

String prepend

```
var str = 'llo!'
str.sub!(/^/, 'He')
say str
```

or

```
var str = 'llo!'
str.prepend!('He')
say str
```

Output:

```
Hello!
```

Strip a set of characters from a string

```
func stripchars(str, char_list) {
    str.tr(char_list, "", "d")
}
```

or:

```
func stripchars(str, char_list) {
    str.chars.grep {|c| !char_list.contains(c)}.join
}
```

Calling the function:

```
say stripchars("She was a soul stripper. She took my heart!", "aei")
```

Output:

```
Sh ws  soul strppr. Sh took my hrt!
```

Strip block comments

For extra credit, it allows the caller to redefine the delimiters.

```
func strip_block_comments(code, beg='/*', end='*/') {
    var re = Regex(beg.escape + '.*?' + end.escape, 's')
    code.gsub(re, '')
}

say strip_block_comments(ARGF.slurp)
```

Strip comments from a string

```
func strip_comment(s) {
    (s - %r'[#;].*').strip
}

[" apples, pears # and bananas",
 " apples, pears ; and bananas",
 " apples, pears "].each { |s|
    say strip_comment(s).dump;
}
```

Output:

```
"apples, pears"
"apples, pears"
"apples, pears"
```

Strip control codes and extended characters from a string

```
var str = "\ba\x00b\n\rc\fd\xc3\x7ffoo"

var letters = str.chars.map{.ord}
say letters.map{.chr}.join.dump

var nocontrols = letters.grep{ (_ > 32) && (_ != 127) }
say nocontrols.map{.chr}.join.dump

var noextended = nocontrols.grep{ _ < 127 }
say noextended.map{.chr}.join.dump
```

Output:

```
"\ba\0b\n\rc\fd\xC3\x7Ffoo"
"abcd\xC3foo"
"abcdfoo"
```

Strip whitespace from a string/Top and tail

```
var s = " \t\v\r\n\ffoo bar \t\v\r\n\f"
say s.strip_beg.dump     # remove leading whitespaces
say s.strip_end.dump     # remove trailing whitespaces
say s.strip.dump         # remove both leading and trailing whitespace
```

Output:

```
"foo bar \t\13\r\n\f"
" \t\13\r\n\ffoo bar"
"foo bar"
```

Strong and weak primes

```
var primes = 10_000_019.primes

var (*strong, *weak, *balanced)

for k in (1 ..^ primes.end) {
    var p = primes[k]

    given((primes[k-1] + primes[k+1])/2) { |x|
        case (x > p) {     weak << p }
        case (x < p) {   strong << p }
        else         { balanced << p }
    }
}

for pr, type, d, c1, c2 in [
    [  strong, 'strong',   36, 1e6, 1e7],
    [    weak, 'weak',     37, 1e6, 1e7],
    [balanced, 'balanced', 28, 1e6, 1e7],
] {
    say ("\nFirst #{d} #{type} primes:\n", pr.first(d).map{.commify}.join(' '))
    say ("Count of #{type} primes <= #{c1.commify}:  ", pr.first_index { _ > 1e6 }.commify)
    say ("Count of #{type} primes <= #{c2.commify}: " , pr.len.commify)
}
```

Output:

```
First 36 strong primes:
11 17 29 37 41 59 67 71 79 97 101 107 127 137 149 163 179 191 197 223 227 239 251 269 277 281 307 311 3
Count of strong primes <= 1,000,000:  37,723
Count of strong primes <= 10,000,000: 320,991

First 37 weak primes:
3 7 13 19 23 31 43 47 61 73 83 89 103 109 113 131 139 151 167 181 193 199 229 233 241 271 283 293 313 3
Count of weak primes <= 1,000,000:  37,780
Count of weak primes <= 10,000,000: 321,750

First 28 balanced primes:
5 53 157 173 211 257 263 373 563 593 607 653 733 947 977 1,103 1,123 1,187 1,223 1,367 1,511 1,747 1,75
Count of balanced primes <= 1,000,000:  2,994
Count of balanced primes <= 10,000,000: 21,837
```

Subleq

```
var memory = ARGV.map{.to_i}
var ip = 0

while (ip.ge(0) && ip.lt(memory.len)) {
    var (a, b, c) = memory[ip, ip+1, ip+2]
    ip += 3
    if (a < 0) {
        memory[b] = STDIN.getc.ord
    }
    elsif (b < 0) {
        print memory[a].chr
    }
    elsif ((memory[b] -= memory[a]) <= 0) {
        ip = c
    }
}
```

Output:

```
$ sidef subleq.sf 15 17 -1 17 -1 -1 16 1 -1 16 3 -1 15 15 0 0 -1 72 101 108 108 111 44 32 119 111 114 1
Hello, world!
```

Subset sum problem

```
var pairs = Hash(
    alliance    => -624, archbishop => -915,
    brute       =>  870, centipede  => -658,
    departure   =>  952, deploy     =>   44,
    elysee      => -326, eradicate  =>  376,
    fiat        =>  170, filmy      => -874,
    infra       => -847, isis       => -982,
    mincemeat   => -880, moresby    =>  756,
    smokescreen =>  423, speakeasy  => -745,
    balm        =>  397, bonnet     =>  452,
    cobol       =>  362, covariate  =>  590,
    diophantine =>  645, efferent   =>   54,
    escritoire  =>  856, exorcism   => -983,
    flatworm    =>  503, gestapo    =>  915,
    lindholm    =>  999, markham    =>  475,
    mycenae     =>  183, plugging   => -266,
    vein        =>  813,
)

var weights = pairs.keys.sort.map{|k| pairs{k} }
var inverse = pairs.flip

for n in (1 .. weights.end) {
    var found = false
    weights.combinations(n, {|*comb|
        if (comb.sum == 0) {
            say "Length #{n}: "+" ".join(inverse{comb...})
            found = true
            break
        }
    })
    found || say "Length #{n}: (none)"
}
```

Output:

```
Length 1: (none)
Length 2: archbishop gestapo
Length 3: centipede markham mycenae
Length 4: alliance balm deploy mycenae
Length 5: alliance brute covariate deploy mincemeat
Length 6: alliance archbishop balm deploy gestapo mycenae
Length 7: alliance archbishop bonnet cobol departure exorcism moresby
Length 8: alliance archbishop balm bonnet fiat flatworm isis lindholm
...
```

Substitution cipher

```
module SubstitutionCipher {

    const key = %c"]kYV}(!7P$n5_0i
R:?jOWtF/=-pe'AD&@r6%ZXs\"v*N[#wSl9zq2^+g;LoB`aGh{3.HIu4fbK)mU8|dMET><,Qc\\C1yxJ"

    func encode(String s) {
        var r = ""
        s.each {|c|
            r += key[c.ord - 32]
        }
        return r
    }

    func decode(String s) {
        var r = ""
        s.each {|c|
            r += (key.first_index { _ == c } + 32 -> chr)
        }
        return r
    }
}


with ("The quick brown fox jumps over the lazy dog, who barks VERY loudly!") { |s|
    var enc = SubstitutionCipher::encode(s)
    var dec = SubstitutionCipher::decode(enc)
    say("Original: ", s, "\n -> Encoded: ", enc, "\n -> Decoded: ", dec)
}
```

Output:

```
Original: The quick brown fox jumps over the lazy dog, who barks VERY loudly!
 -> Encoded: 2bu]E,KHm].Tdc|]4d\]),8M>]dQuT]<bu]U31C]Idf_]cbd].3Tm>]+ZzL]Ud,IUCk
 -> Decoded: The quick brown fox jumps over the lazy dog, who barks VERY loudly!
```

Substring

```
var str = 'abcdefgh'
var n = 2
var m = 3
say str.substr(n, m)                    #=> cde
say str.substr(n)                       #=> cdefgh
say str.substr(0, -1)                   #=> abcdefg
say str.substr(str.index('d'), m)       #=> def
say str.substr(str.index('de'), m)      #=> def
```

Substring/Top and tail

Strip any characters:

```
say "knight".substr(1)        # strip first character
say "socks".substr(0, -1)     # strip last character
say "brooms".substr(1, -1)    # strip both first and last characters
say "与今令".substr(1, -1)     # => 今
```

Output:

```
night
sock
room
今
```

Strip graphemes:

```
var gstr = "J\x{332}o\x{332}s\x{332}e\x{301}\x{332}"
say gstr-/^\X/                    # strip first grapheme
say gstr-/\X\z/                   # strip last grapheme
say gstr.sub(/^\X/).sub(/\X\z/)   # strip both first and last graphemes
```

Output:

```
osé̲
Jos
os
```

Substring primes

Generic solution for any base >= 2.

```
func split_at_indices(array, indices) {

    var parts = []
    var i = 0

    for j in (indices) {
        parts << array.slice(i, j)
        i = j+1
    }

    parts
}

func consecutive_partitions(array, callback) {
    for k in (0..array.len) {
        combinations(array.len, k, {|*indices|
            var t = split_at_indices(array, indices)
            if (t.sum_by{.len} == array.len) {
                callback(t)
            }
        })
    }
}

func is_substring_prime(digits, base) {

    for k in (^digits) {
        digits.first(k+1).digits2num(base).is_prime || return false
    }

    consecutive_partitions(digits, {|part|
        part.all { .digits2num(base).is_prime } || return false
    })

    return true
}

func generate_from_prefix(p, base, digits) {

    var seq = [p]

    for d in (digits) {
        var t = [d, p...]
        if (is_prime(t.digits2num(base)) && is_substring_prime(t, base)) {
            seq << __FUNC__(t, base, digits)...
        }
    }

    return seq
}

func substring_primes(base) {     # finite sequence for each base >= 2

    var prime_digits = (base-1 -> primes)   # prime digits < base

    prime_digits.map  {|p| generate_from_prefix([p], base, prime_digits)... }\
                .map  {|t| digits2num(t, base) }\
                .sort
}

for base in (2..20) {
    say "base = #{base}: #{substring_primes(base)}"
}
```

Output:

```
base = 2: []
base = 3: [2]
base = 4: [2, 3, 11]
base = 5: [2, 3, 13, 17, 67]
base = 6: [2, 3, 5, 17, 23]
base = 7: [2, 3, 5, 17, 19, 23, 37]
base = 8: [2, 3, 5, 7, 19, 23, 29, 31, 43, 47, 59, 61, 157, 239, 251, 349, 379, 479, 491]
base = 9: [2, 3, 5, 7, 23, 29, 47]
base = 10: [2, 3, 5, 7, 23, 37, 53, 73, 373]
base = 11: [2, 3, 5, 7, 29, 79]
base = 12: [2, 3, 5, 7, 11, 29, 31, 41, 43, 47, 67, 71, 89, 137, 139, 359, 499, 503, 521, 569, 571, 809
base = 13: [2, 3, 5, 7, 11, 29, 31, 37, 41, 67, 379]
base = 14: [2, 3, 5, 7, 11, 13, 31, 41, 47, 53, 73, 83, 101, 103, 109, 157, 167, 193, 439, 661, 1033, 2
base = 15: [2, 3, 5, 7, 11, 13, 37, 41, 43, 47, 107, 167, 197, 557, 617, 647]
base = 16: [2, 3, 5, 7, 11, 13, 37, 43, 53, 59, 61, 83, 179, 181, 211, 691, 947, 3389]
base = 17: [2, 3, 5, 7, 11, 13, 37, 41, 47, 53, 223, 631]
base = 18: [2, 3, 5, 7, 11, 13, 17, 41, 43, 47, 53, 59, 61, 67, 71, 97, 101, 103, 107, 131, 137, 139, 2
base = 19: [2, 3, 5, 7, 11, 13, 17, 41, 43, 59, 97, 211]
base = 20: [2, 3, 5, 7, 11, 13, 17, 19, 43, 47, 53, 59, 67, 71, 73, 79, 103, 107, 113, 151, 157, 223, 2
```

Subtractive generator

```
class SubRandom(seed, state=[]) {

    const mod = 1_000_000_000

    method init {
        var s = [seed % mod, 1]
        53.times {
            s.append((s[-2] - s[-1]) % mod)
        }
        state = s.range.map {|i| s[(34 + 34*i) % 55] }
        165.times { self.subrand }
    }

    method subrand {
        var x = ((state.shift - state[-24]) % mod)
        state.append(x)
        return x
    }
}

var r = SubRandom(292929)
10.times { say r.subrand }
```

Output:

```
467478574
512932792
539453717
20349702
615542081
378707948
933204586
824858649
506003769
380969305
```

Successive prime differences

```
var limit  = 1e6
var primes = limit.primes

say "Groups of successive primes <= #{limit.commify}:"

for diffs in [[2], [1], [2,2], [2,4], [4,2], [6,4,2]] {

    var groups = []
    primes.each_cons(diffs.len+1, {|*group|
        if (group.map_cons(2, {|a,b| b-a}) == diffs) {
            groups << group
        }
    })

    say ("...for differences #{diffs}, there are #{groups.len} groups, where ",
         "the first group = #{groups.first} and the last group = #{groups.last}")
}
```

Output:

```
Groups of successive primes <= 1,000,000:
...for differences [2], there are 8169 groups, where the first group = [3, 5] and the last group = [999
...for differences [1], there are 1 groups, where the first group = [2, 3] and the last group = [2, 3]
...for differences [2, 2], there are 1 groups, where the first group = [3, 5, 7] and the last group = [
...for differences [2, 4], there are 1393 groups, where the first group = [5, 7, 11] and the last group
...for differences [4, 2], there are 1444 groups, where the first group = [7, 11, 13] and the last grou
...for differences [6, 4, 2], there are 306 groups, where the first group = [31, 37, 41, 43] and the la
```

Sudoku

```
func check(i, j) is cached {
    var (id, im) = i.divmod(9)
    var (jd, jm) = j.divmod(9)

    jd == id && return true
    jm == im && return true

    (id//3 == jd//3) &&
    (jm//3 == im//3)
}

func solve(grid) {
    for i in ^grid {
        grid[i] && next
        var t = [grid[{|j| check(i, j) }.grep(^grid)]].freq

        { |k|
            t.has_key(k) && next
            grid[i] = k
            solve(grid)
        } << 1..9

        grid[i] = 0
        return nil
    }

    for i in ^grid {
        print "#{grid[i]} "
        print " "  if (3  -> divides(i+1))
        print "\n" if (9  -> divides(i+1))
        print "\n" if (27 -> divides(i+1))
    }
}

var grid = %i(
    5 3 0  0 2 4  7 0 0
    0 0 2  0 0 0  8 0 0
    1 0 0  7 0 3  9 0 2

    0 0 8  0 7 2  0 4 9
    0 2 0  9 8 0  0 7 0
    7 9 0  0 0 0  0 8 0

    0 0 0  0 3 0  5 0 6
    9 6 0  0 1 0  3 0 0
    0 5 0  6 9 0  0 1 0
)

solve(grid)
```

Output:

```
5 3 9  8 2 4  7 6 1
6 7 2  1 5 9  8 3 4
1 8 4  7 6 3  9 5 2

3 1 8  5 7 2  6 4 9
4 2 5  9 8 6  1 7 3
7 9 6  3 4 1  2 8 5

8 4 1  2 3 7  5 9 6
9 6 7  4 1 5  3 2 8
2 5 3  6 9 8  4 1 7
```

Suffix tree

```
func suffix_tree(Str t) {
    suffix_tree(^t.len -> map { t.substr(_) })
}

func suffix_tree(Arr a {.len == 1}) {
    Hash(a[0] => nil)
}

func suffix_tree(Arr a) {
    var h = Hash()
    for k,v in (a.group_by { .char(0) }) {
        var subtree = suffix_tree(v.map { .substr(1) })
        var subkeys = subtree.keys
        if (subkeys.len == 1) {
            var subk = subkeys[0]
            h{k + subk} = subtree{subk}
        }
        else {
            h{k} = subtree
        }
    }
    return h
}

say suffix_tree('banana$')
```

Output:

```
Hash(
    "$" => nil,
    "a" => Hash(
        "$" => nil,
        "na" => Hash(
            "$" => nil,
            "na$" => nil
        )
    ),
    "banana$" => nil,
    "na" => Hash(
        "$" => nil,
        "na$" => nil
    )
)
```

Sum and product of an array

Using built-in methods:

```
var ary = [1, 2, 3, 4, 5]
say ary.sum                 # => 15
say ary.prod                # => 120
```

Alternatively, using hyper-operators:

```
var ary = [1, 2, 3, 4, 5]
say ary«+»                  # => 15
say ary«*»                  # => 120
```

Sum and product puzzle

```
func grep_uniq(a, by) { a.group_by{ .(by) }.values.grep{.len == 1}.map{_[0]} }
func sums     (n)     { 2 .. n//2 -> map {|i| [i, n-i] } }

var pairs = {|i| ([i] ~X (i+1 .. 98))... }.map(2..97)

var p_uniq = Hash()
p_uniq{grep_uniq(pairs, :prod).map { .to_s }...} = ()

var s_pairs = pairs.grep {|p| sums(p.sum).all { !p_uniq.contains(.to_s) } }
var p_pairs = grep_uniq(s_pairs, :prod)
var f_pairs = grep_uniq(p_pairs, :sum)

f_pairs.each { |p| printf("X = %d, Y = %d\n", p...) }
```

Output:

```
X = 4, Y = 13
```

Sum digits of an integer

```
func Σ(String str, base=36) {
    str.chars.map{ Num(_, base) }.sum
}

<1 1234 1020304 fe f0e DEADBEEF>.each { |n|
    say "Σ(#{n}) = #{Σ(n)}"
}
```

Output:

```
Σ(1) = 1
Σ(1234) = 10
Σ(1020304) = 10
Σ(fe) = 29
Σ(f0e) = 29
Σ(DEADBEEF) = 104
```

Sum multiples of 3 and 5

```
func sumMul(n, f) {
    var m = int((n - 1) / f)
    f * m * (m + 1) / 2
}

func sum35(n) {
    sumMul(n, 3) + sumMul(n, 5) - sumMul(n, 15)
}

for i in (1..20) {
    printf("%2s:%22s %s\n", i, 10**i, sum35(10**i))
}
```

Output:

```
 1:                    10 23
 2:                   100 2318
 3:                  1000 233168
 4:                 10000 23331668
 5:                100000 2333316668
 6:               1000000 233333166668
 7:              10000000 23333331666668
 8:             100000000 2333333316666668
 9:            1000000000 233333333166666668
10:           10000000000 2333333333166666668
11:          100000000000 23333333333166666668
12:         1000000000000 233333333333166666668
13:        10000000000000 2333333333333166666668
14:       100000000000000 23333333333333166666668
15:      1000000000000000 233333333333333166666668
16:     10000000000000000 2333333333333333166666668
17:    100000000000000000 23333333333333333166666668
18:   1000000000000000000 233333333333333333166666668
19:  10000000000000000000 2333333333333333333166666668
20: 100000000000000000000 23333333333333333333166666668
```

Sum of a series

```
say sum(1..1000, {|n| 1 / n**2 })
```

Alternatively, using the *reduce{}* method:

```
say (1..1000 -> reduce { |a,b| a + (1 / b**2) })
```

Output:

```
1.643934566681559803139058023822215589652103446494
```

Sum of divisors

```
1..100 -> map { .sigma }.say
```

Output:

```
[1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24, 31, 18, 39, 20, 42, 32, 36, 24, 60, 31, 42, 40,
```

Sum of first n cubes

```
0..49 -> map { .faulhaber_sum(3) }.slices(5).each { .join(' ').say }
```

Output:

```
0 1 9 36 100
225 441 784 1296 2025
3025 4356 6084 8281 11025
14400 18496 23409 29241 36100
44100 53361 64009 76176 90000
105625 123201 142884 164836 189225
216225 246016 278784 314721 354025
396900 443556 494209 549081 608400
672400 741321 815409 894916 980100
1071225 1168561 1272384 1382976 1500625
```

Sum of primes in odd positions is prime

```
var sum = 0
1e3.primes.each_kv {|k,v|
    if (k+1 -> is_odd) {
        sum += v
        say "#{k+1} #{v} #{sum}" if sum.is_prime
    }
}
```

Output:

```
1 2 2
3 5 7
11 31 89
27 103 659
35 149 1181
67 331 5021
91 467 9923
95 499 10909
99 523 11941
119 653 17959
143 823 26879
```

Sum of square and cube digits of an integer are primes

```
1..99 -> grep { .square.digits_sum.is_prime && .cube.digits_sum.is_prime }.say
```

Output:

```
[16, 17, 25, 28, 34, 37, 47, 52, 64]
```

Sum of squares

```
func sum_of_squares(vector) {
    var sum = 0
    vector.each { |n| sum += n**2 }
    return sum
}

say sum_of_squares([])          # 0
say sum_of_squares([1,2,3])     # 14
```

Sum of the digits of n is substring of n

```
var upto = 1000
var base = 10

var list = (^upto -> grep {
    .digits(base).contains(.sumdigits(base).digits(base)...)
})

say "Numbers under #{upto} whose sum of digits is a substring of themselves:"

list.each_slice(8, {|*a|
    say a.map { '%3s' % _ }.join(' ')
})

say "\n#{list.len} such numbers found."
```

Output:

```
Numbers under 1000 whose sum of digits is a substring of themselves:
  0   1   2   3   4   5   6   7
  8   9  10  20  30  40  50  60
 70  80  90 100 109 119 129 139
149 159 169 179 189 199 200 300
400 500 600 700 800 900 910 911
912 913 914 915 916 917 918 919

48 such numbers found.
```

Sum of two adjacent numbers are primes

```
var wanted = (1..Inf -> lazy.map  {|n| [n, n+1, n+(n+1)] }\
                           .grep { .tail.is_prime })

wanted.first(20).each_2d {|a,b,c|
    printf("%2d + %2d = %2d\n", a,b,c)
}
```

Output:

```
 1 +  2 =  3
 2 +  3 =  5
 3 +  4 =  7
 5 +  6 = 11
 6 +  7 = 13
 8 +  9 = 17
 9 + 10 = 19
11 + 12 = 23
14 + 15 = 29
15 + 16 = 31
18 + 19 = 37
20 + 21 = 41
21 + 22 = 43
23 + 24 = 47
26 + 27 = 53
29 + 30 = 59
30 + 31 = 61
33 + 34 = 67
35 + 36 = 71
36 + 37 = 73
```

Sum to 100

```
func gen_expr() is cached {
    var x = ['-', '']
    var y = ['+', '-', '']

    gather {
        cartesian([x,y,y,y,y,y,y,y,y], {|a,b,c,d,e,f,g,h,i|
            take("#{a}1#{b}2#{c}3#{d}4#{e}5#{f}6#{g}7#{h}8#{i}9")
        })
    }
}

func eval_expr(expr) is cached {
    expr.scan(/([-+]?\d+)/).sum_by { Num(_) }
}

func sum_to(val) {
    gen_expr().grep { eval_expr(_) == val }
}

func max_solve() {
    gen_expr().grep     { eval_expr(_) >= 0 } \
              .group_by { eval_expr(_)      } \
              .max_by   {|_,v| v.len        }
}

func min_solve() {
    var h = gen_expr().group_by { eval_expr(_) }
    for i in (0..Inf) { h.exists(i) || return i }
}

func highest_sums(n=10) {
    gen_expr().map { eval_expr(_) }.uniq.sort.reverse.first(n)
}

sum_to(100).each { say "100 = #{_}" }

var (n, solutions) = max_solve()...
say "Sum of #{n} has the maximum number of solutions: #{solutions.len}"
say "Lowest positive sum that can't be expressed : #{min_solve()}"
say "Highest sums: #{highest_sums()}"
```

Output:

```
100 = -1+2-3+4+5+6+78+9
100 = 1+2+3-4+5+6+78+9
100 = 1+2+34-5+67-8+9
100 = 1+23-4+5+6+78-9
100 = 1+23-4+56+7+8+9
100 = 12+3+4+5-6-7+89
100 = 12+3-4+5+67+8+9
100 = 12-3-4+5-6+7+89
100 = 123+4-5+67-89
100 = 123+45-67+8-9
100 = 123-4-5-6-7+8-9
100 = 123-45-67+89
Sum of 9 has the maximum number of solutions: 46
Lowest positive sum that can't be expressed : 211
Highest sums: [123456789, 23456790, 23456788, 12345687, 12345669, 3456801, 3456792, 3456790, 3456788, 3
```

Summarize primes

```
1000.primes.map_reduce {|a,b| a + b }.map_kv {|k,v|
    [k+1, prime(k+1), v]
}.grep { .tail.is_prime }.prepend(
    ['count', 'prime', 'sum']
).each_2d {|n,p,s|
    printf("%5s %6s %8s\n", n, p, s)
}
```

Output:

```
count  prime      sum
    1      2        2
    2      3        5
    4      7       17
    6     13       41
   12     37      197
   14     43      281
   60    281     7699
   64    311     8893
   96    503    22039
  100    541    24133
  102    557    25237
  108    593    28697
  114    619    32353
  122    673    37561
  124    683    38921
  130    733    43201
  132    743    44683
  146    839    55837
  152    881    61027
  158    929    66463
  162    953    70241
```

Summation of primes

Built-in:

```
say sum_primes(2e6)  #=> 142913828922
```

Linear algorithm:

```
func sum_primes(limit) {
    var sum = 0
    for (var p = 2; p < limit; p.next_prime!) {
        sum += p
    }
    return sum
}

say sum_primes(2e6)
```

Sublinear algorithm:

```
func sum_of_primes(n) {

    return 0 if (n <= 1)

    var r = n.isqrt
    var V = (1..r -> map {|k| idiv(n,k) })
    V << range(V.last-1, 1, -1).to_a...

    var S = Hash(V.map{|k| (k, polygonal(k,3)) }...)

    for p in (2..r) {
        S{p} > S{p-1} || next
        var sp = S{p-1}
        var p2 = p*p
        V.each {|v|
            break if (v < p2)
            S{v} -= p*(S{idiv(v,p)} - sp)
        }
    }

    return S{n}-1
}

say sum_of_primes(2e6)
```

Output:

```
142913828922
```

Sunflower fractal

```
require('Imager')

func draw_sunflower(seeds=3000) {
    var img = %O<Imager>.new(
        xsize => 400,
        ysize => 400,
    )

    var c = (sqrt(1.25) + 0.5)
    { |i|
        var r = (i**c / seeds)
        var θ = (2 * Num.pi * c * i)
        var x = (r * sin(θ) + 200)
        var y = (r * cos(θ) + 200)
        img.circle(x => x, y => y, r => i/(5*seeds))
    } * seeds

    return img
}

var img = draw_sunflower()
img.write(file => "sunflower.png")
```

Sunflower fractal

Super-d numbers

```
func super_d(d) {
    var D = Str(d)*d
    1..Inf -> lazy.grep {|n| Str(d * n**d).contains(D) }
}

for d in (2..8) {
    say ("#{d}: ", super_d(d).first(10))
}
```

Output:

```
2: [19, 31, 69, 81, 105, 106, 107, 119, 127, 131]
3: [261, 462, 471, 481, 558, 753, 1036, 1046, 1471, 1645]
4: [1168, 4972, 7423, 7752, 8431, 10267, 11317, 11487, 11549, 11680]
5: [4602, 5517, 7539, 12955, 14555, 20137, 20379, 26629, 32767, 35689]
6: [27257, 272570, 302693, 323576, 364509, 502785, 513675, 537771, 676657, 678146]
7: [140997, 490996, 1184321, 1259609, 1409970, 1783166, 1886654, 1977538, 2457756, 2714763]
8: [185423, 641519, 1551728, 1854230, 6415190, 12043464, 12147605, 15517280, 16561735, 18542300]
```

Superellipse

```
const (
    a = 200,
    b = 200,
    n = 2.5,
)

# y in terms of x
func y(x) { b * (1 - abs(x/a)**n -> root(n)) -> int }

func pline(q) {
    <<-"EOT";
    <polyline points="#{q.join(' ')}"
    style="fill:none; stroke:black; stroke-width:3" transform="translate(#{a}, #{b})" />
    EOT
}

# Generate an SVG image
say <<-"EOT"
    <?xml version="1.0" standalone="no"?>
    <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
    <svg height="#{b*2}" width="#{a*2}" version="1.1" xmlns="http://www.w3.org/2000/svg">
    EOT

# find point pairs for one quadrant
var q = { |x| (x, y(x)) }.map(0..200 `by` 2)

[
    pline(q),
    pline(q »*« [ 1,-1]), # flip and mirror
    pline(q »*« [-1,-1]), # for the other
    pline(q »*« [-1, 1]), # three quadrants
].each { .print }

say '</svg>'
```

Superpermutation minimisation

```
for len in (1..8) {
    var (pre="", post="")
    @^len -> permutations {|*p|
        var t = p.join
        post.append!(t) if !post.contains(t)
        pre.prepend!(t) if !pre.contains(t)
    }
    printf("%2d: %8d %8d\n", len, pre.len, post.len)
}
```

Output:

```
 1:        1        1
 2:        4        4
 3:       12       15
 4:       48       64
 5:      240      325
 6:     1440     1956
 7:    10080    13699
 8:    80640   109600
```

Sutherland-Hodgman polygon clipping

```
class Point(x, y) {
    method to_s {
        "(#{'%.2f' % x}, #{'%.2f' % y})"
    }
}

func sutherland_hodgman(subjectPolygon, clipPolygon) {
  var inside = { |cp1, cp2, p|
    ((cp2.x-cp1.x)*(p.y-cp1.y)) > ((cp2.y-cp1.y)*(p.x-cp1.x))
  }

  var intersection = { |cp1, cp2, s, e|
    var (dcx, dcy) = (cp1.x-cp2.x, cp1.y-cp2.y)
    var (dpx, dpy) = (s.x-e.x, s.y-e.y)
    var n1 = (cp1.x*cp2.y - cp1.y*cp2.x)
    var n2 = (s.x*e.y - s.y*e.x)
    var n3 = (1 / (dcx*dpy - dcy*dpx))
    Point((n1*dpx - n2*dcx) * n3, (n1*dpy - n2*dcy) * n3)
  }

  var outputList = subjectPolygon
  var cp1 = clipPolygon.last
  for cp2 in clipPolygon {
    var inputList = outputList
    outputList = []
    var s = inputList.last
    for e in inputList {
      if (inside(cp1, cp2, e)) {
        outputList << intersection(cp1, cp2, s, e) if !inside(cp1, cp2, s)
        outputList << e
      }
      elsif(inside(cp1, cp2, s)) {
        outputList << intersection(cp1, cp2, s, e)
      }
      s = e
    }
    cp1 = cp2
  }
  outputList
}

var subjectPolygon = [
    [50,  150], [200,  50], [350, 150], [350, 300],
    [250, 300], [200, 250], [150, 350], [100, 250],
    [100, 200]
].map{|pnt| Point(pnt...) }

var clipPolygon = [
    [100, 100], [300, 100],
    [300, 300], [100, 300]
].map{|pnt| Point(pnt...) }

sutherland_hodgman(subjectPolygon, clipPolygon).each { .say }
```

Output:

```
(100.00, 116.67)
(125.00, 100.00)
(275.00, 100.00)
(300.00, 116.67)
(300.00, 300.00)
(250.00, 300.00)
(200.00, 250.00)
(175.00, 300.00)
(125.00, 300.00)
(100.00, 250.00)
```

Sylvester's sequence

```
func sylvester_sequence(n) {
    1..n -> reduce({|a| a*(a-1) + 1 }, 2)
}

say "First 10 terms in Sylvester's sequence:"
10.of(sylvester_sequence).each_kv{|k,v| '%2s: %s' % (k,v) -> say }

say "\nSum of reciprocals of first 10 terms: "
say 10.of(sylvester_sequence).sum {|n| 1/n }.as_dec(230)
```

Output:

```
First 10 terms in Sylvester's sequence:
 0: 2
 1: 3
 2: 7
 3: 43
 4: 1807
 5: 3263443
 6: 10650056950807
 7: 113423713055421844361000443
 8: 12864938683278671740537145998360961546653259485195807
 9: 165506647324519964198468195444439180017513152706377497841851388766535868639572406808911988131737645

Sum of reciprocals of first 10 terms:
0.99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
```

Symmetric difference

```
var a = ["John", "Serena", "Bob", "Mary", "Serena"]
var b = ["Jim", "Mary", "John", "Jim", "Bob"]
say (a ^ b -> uniq)
```

Output:

```
["Jim", "Serena"]
```

System time

```
# textual
say Time.local.ctime         # => Thu Mar 19 15:10:41 2015

# epoch time
say Time.sec                 # => 1426770641

# epoch time with fractional seconds
say Time.micro_sec           # => 1426770641.68409
```

Table creation/Postal addresses

```
require('DBI');

var db = %s'DBI'.connect('DBI:mysql:database:server','login','password');

var statment = <<'EOF';
CREATE TABLE `Address` (
    `addrID`       int(11)     NOT NULL   auto_increment,
    `addrStreet`   varchar(50) NOT NULL   default '',
    `addrCity`     varchar(25) NOT NULL   default '',
    `addrState`    char(2)     NOT NULL   default '',
    `addrZIP`      char(10)    NOT NULL   default '',
    PRIMARY KEY (`addrID`)
);
EOF

var exec = db.prepare(statment);
exec.execute;
```

Take notes on the command line

```
var file = %f'notes.txt'

if (ARGV.len > 0) {
    var fh = file.open_a
    fh.say(Time.local.ctime + "\n\t" + ARGV.join(" "))
    fh.close
} else {
    var fh = file.open_r
    fh && fh.each { .say }
}
```

Output:

```
$ sidef notes.sf Test 1
$ sidef notes.sf Test 2
$ sidef notes.sf
Sat Sep 12 00:19:36 2015
        Test 1
Sat Sep 12 00:19:37 2015
        Test 2
```

Tarjan

```
func tarjan (k) {

    var(:onstack, :index, :lowlink, *stack, *connected)

    func strong_connect (vertex, i=0) {

         index{vertex}   = i
         lowlink{vertex} = i+1
         onstack{vertex} = true
         stack << vertex

         for connection in (k{vertex}) {
             if (index{connection} == nil) {
                 strong_connect(connection, i+1)
                 lowlink{vertex} `min!` lowlink{connection}
             }
             elsif (onstack{connection}) {
                 lowlink{vertex} `min!` index{connection}
             }
        }

        if (lowlink{vertex} == index{vertex}) {
            var *node
            do {
                node << stack.pop
                onstack{node.tail} = false
            } while (node.tail != vertex)
            connected << node
        }
    }

    { strong_connect(_) if !index{_} } << k.keys

    return connected
}

var tests = [
    Hash(
         0 => <1>,
         1 => <2>,
         2 => <0>,
         3 => <1 2 4>,
         4 => <3 5>,
         5 => <2 6>,
         6 => <5>,
         7 => <4 6 7>,
    ),
    Hash(
        :Andy => <Bart>,
        :Bart => <Carl>,
        :Carl => <Andy>,
        :Dave => <Bart Carl Earl>,
        :Earl => <Dave Fred>,
        :Fred => <Carl Gary>,
        :Gary => <Fred>,
        :Hank => <Earl Gary Hank>,
    )
]

tests.each {|t|
    say ("Strongly connected components: ", tarjan(t).map{.sort}.sort)
}
```

Output:

```
Strongly connected components: [["0", "1", "2"], ["3", "4"], ["5", "6"], ["7"]]
Strongly connected components: [["Andy", "Bart", "Carl"], ["Dave", "Earl"], ["Fred", "Gary"], ["Hank"]]
```

Tau function

Built-in:

```
say { .sigma0 }.map(1..100).join(' ')
```

Output:

```
1 2 2 3 2 4 2 4 3 4 2 6 2 4 4 5 2 6 2 6 4 4 2 8 3 4 4 6 2 8 2 6 4 4 4 9 2 4 4 8 2 8 2 6 6 4 2 10 3 6 4
```

Tau number

```
func is_tau_number(n) {
    n % n.sigma0 == 0
}

say is_tau_number.first(100).join(' ')
```

Output:

```
1 2 8 9 12 18 24 36 40 56 60 72 80 84 88 96 104 108 128 132 136 152 156 180 184 204 225 228 232 240 248
```

Taxicab numbers

```
var (start=1, end=25) = ARGV.map{.to_i}...

func display (h, start, end) {
    var i = start
    for n in [h.grep {|_,v| v.len > 1 }.keys.sort_by{.to_i}[start-1 .. end-1]] {
        printf("%4d %10d  =>\t%s\n", i++, n,
            h{n}.map{ "%4d³ + %-s" % (.first, "#{.last}³") }.join(",\t"))
    }
}

var taxi = Hash()
var taxis = 0
var terminate = 0

for c1 (1..Inf) {
    if (0<terminate && terminate<c1) {
        display(taxi, start, end)
        break
    }
    var c = c1**3
    for c2 (1..c1) {
        var this = (c2**3 + c)
        taxi{this} := [] << [c2, c1]
        ++taxis if (taxi{this}.len == 2)
        if (taxis==end && !terminate) {
            terminate = taxi.grep{|_,v| v.len > 1 }.keys.map{.to_i}.max.root(3)
        }
    }
}
```

Output:

```
   1       1729  =>        9³ + 10³,       1³ + 12³
   2       4104  =>        9³ + 15³,       2³ + 16³
   3      13832  =>       18³ + 20³,       2³ + 24³
   4      20683  =>       19³ + 24³,      10³ + 27³
   5      32832  =>       18³ + 30³,       4³ + 32³
   6      39312  =>       15³ + 33³,       2³ + 34³
   7      40033  =>       16³ + 33³,       9³ + 34³
   8      46683  =>       27³ + 30³,       3³ + 36³
   9      64232  =>       26³ + 36³,      17³ + 39³
  10      65728  =>       31³ + 33³,      12³ + 40³
  11     110656  =>       36³ + 40³,       4³ + 48³
  12     110808  =>       27³ + 45³,       6³ + 48³
  13     134379  =>       38³ + 43³,      12³ + 51³
  14     149389  =>       29³ + 50³,       8³ + 53³
  15     165464  =>       38³ + 48³,      20³ + 54³
  16     171288  =>       24³ + 54³,      17³ + 55³
  17     195841  =>       22³ + 57³,       9³ + 58³
  18     216027  =>       22³ + 59³,       3³ + 60³
  19     216125  =>       45³ + 50³,       5³ + 60³
  20     262656  =>       36³ + 60³,       8³ + 64³
  21     314496  =>       30³ + 66³,       4³ + 68³
  22     320264  =>       32³ + 66³,      18³ + 68³
  23     327763  =>       51³ + 58³,      30³ + 67³
  24     373464  =>       54³ + 60³,       6³ + 72³
  25     402597  =>       56³ + 61³,      42³ + 69³
```

With passed parameters 2000 and 2006:

Output:

```
2000 1671816384  =>      940³ + 944³,    428³ + 1168³
2001 1672470592  =>      632³ + 1124³,    29³ + 1187³
2002 1673170856  =>      828³ + 1034³,   458³ + 1164³
2003 1675045225  =>      744³ + 1081³,   522³ + 1153³
2004 1675958167  =>      711³ + 1096³,   492³ + 1159³
2005 1676926719  =>      714³ + 1095³,    63³ + 1188³
2006 1677646971  =>      891³ + 990³,     99³ + 1188³
```

Temperature conversion

```
var scale = Hash(
    Celcius    => Hash(factor => 1   , offset => -273.15 ),
    Rankine    => Hash(factor => 1.8, offset =>    0     ),
    Fahrenheit => Hash(factor => 1.8, offset => -459.67 ),
)

var kelvin = read("Enter a temperature in Kelvin: ", Number)
kelvin >= 0 || die "No such temperature!"

scale.keys.sort.each { |key|
    printf("%12s:%8.2f\n", key, kelvin*scale{key}{:factor} + scale{key}{:offset})
}
```

Output:

```
Enter a temperature in Kelvin: 256
     Celcius:  -17.15
  Fahrenheit:    1.13
     Rankine:  460.80
```

Terminal control/Clear the screen

Using a cached-function:

```
func clear { print(static x = `clear`) }
clear()
```

Directly invoking the `clear` command:

```
Sys.run('clear')
```

Alternatively, without reliance on the command line:

```
print "\e[3J\e[H\e[2J"
```

Terminal control/Coloured text

```
var a = frequire('Term::ANSIColor')

say a.colored('RED ON WHITE', 'bold red on_white')
say a.colored('GREEN', 'bold green')
say a.colored('BLUE ON YELLOW', 'bold blue on_yellow')
say a.colored('MAGENTA', 'bold magenta')
say a.colored('CYAN ON RED', 'bold cyan on_red')
say a.colored('YELLOW', 'bold yellow')
```

Terminal control/Dimensions

```
var stty = `stty -a`
var lines = stty.match(/\brows\h+(\d+)/)
var cols  = stty.match(/\bcolumns\h+(\d+)/)
say "#{lines} #{cols}"
```

Output:

```
24 80
```

Terminal control/Display an extended character

```
say '£'
say "\x{FFE1}"
say "\N{FULLWIDTH POUND SIGN}"
say 0xffe1.chr
```

Terminal control/Preserve screen

```
print "\e[?1049h\e[H"
say "Alternate buffer!"

3.downto(1).each { |i|
    say "Going back in: #{i}"
    Sys.sleep(1)
}

print "\e[?1049l"
```

Terminal control/Ringing the terminal bell

```
print "\a"
```

Terminal control/Unicode output

```
if (/\bUTF-?8/i ~~ [ENV{"LC_ALL","LC_CTYPE","LANG"}]) {
    say "△"
} else {
    die "Terminal can't handle UTF-8.\n"
}
```

Test integerness

```
func is_int (n, tolerance=0) {
    !!(abs(n.real.round + n.imag - n) <= tolerance)
}

%w(25.000000 24.999999 25.000100 -2.1e120 -5e-2 Inf NaN 5.0+0.0i 5-5i).each {|s|
    var n = Number(s)
    printf("%-10s  %-8s  %-5s\n", s,
        is_int(n),
        is_int(n, tolerance: 0.00001))
}
```

Output:

```
25.000000   true      true
24.999999   false     true
25.000100   false     false
-2.1e120    true      true
-5e-2       false     false
Inf         false     false
NaN         false     false
5.0+0.0i    true      true
5-5i        false     false
```

Text between

Uses /^/ and /$/ as start and end delimiters. Additionally, the start and end delimiters can be regular expressions.

```
func text_between (text, beg, end) {

    beg.escape! if beg.kind_of(String)
    end.escape! if end.kind_of(String)

    Regex("#{beg}(.*?)(?:#{end}|\\z)", 's').match(text)[0] \\ ""
}

var tests = [
    Hash(
        text  => "Hello Rosetta Code world",
        start => "Hello ",
        end   => " world",
        out   => "Rosetta Code",
    ),
    Hash(
        text  => "Hello Rosetta Code world",
        start => /^/,
        end   => " world",
        out   => "Hello Rosetta Code",
```

```
    ),
    Hash(
        text  => "Hello Rosetta Code world",
        start => "Hello ",
        end   => /$/,
        out   => "Rosetta Code world",
    ),
    Hash(
        text  => "</div><div style=\"chinese\">你好嗎</div>",
        start => "<div style=\"chinese\">",
        end   => "</div>",
        out   => "你好嗎",
    ),
    Hash(
        text  => "<text>Hello <span>Rosetta Code</span> world</text><table style=\"myTable\">",
        start => "<text>",
        end   => "<table>",
        out   => "Hello <span>Rosetta Code</span> world</text><table style=\"myTable\">",
    ),
    Hash(
        text  => "<table style=\"myTable\"><tr><td>hello world</td></tr></table>",
        start => "<table>",
        end   => "</table>",
        out   => "",
    ),
    Hash(
        text  => "The quick brown fox jumps over the lazy other fox",
        start => "quick ",
        end   => " fox",
        out   => "brown",
    ),
    Hash(
        text  => "One fish two fish red fish blue fish",
        start => "fish ",
        end   => " red",
        out   => "two fish",
    ),
    Hash(
        text  => "FooBarBazFooBuxQuux",
        start => "Foo",
        end   => "Foo",
        out   => "BarBaz",
    ),
]

tests.each { |t|
    var r = text_between(t{:text}, t{:start}, t{:end})
    assert_eq(t{:out}, r)
    say "text_between(#{t{:text}.dump}, #{t{:start}.dump}, #{t{:end}.dump}) = #{r.dump}"
}
```

Output:

```
text_between("Hello Rosetta Code world", "Hello ", " world") = "Rosetta Code"
text_between("Hello Rosetta Code world", /^/, " world") = "Hello Rosetta Code"
text_between("Hello Rosetta Code world", "Hello ", /$/) = "Rosetta Code world"
text_between("</div><div style=\"chinese\">\x{4F60}\x{597D}\x{55CE}</div>", "<div style=\"chinese\">",
text_between("<text>Hello <span>Rosetta Code</span> world</text><table style=\"myTable\">", "<text>", "
text_between("<table style=\"myTable\"><tr><td>hello world</td></tr></table>", "<table>", "</table>") =
text_between("The quick brown fox jumps over the lazy other fox", "quick ", " fox") = "brown"
text_between("One fish two fish red fish blue fish", "fish ", " red") = "two fish"
text_between("FooBarBazFooBuxQuux", "Foo", "Foo") = "BarBaz"
```

Text processing/1

```
var gaps = [];
var previous = :valid;

ARGF.each { |line|
    var (date, *readings) = line.words...;
    var valid = [];
    var hour = 0;
    for reading, flag in readings.map{.to_n}.slices(2) {
        if (flag > 0) {
            valid << reading;
            if (previous == :invalid) {
                gaps[-1]{:end} = "#{date} #{hour}:00";
                previous = :valid;
            }
        }
        else {
            if (previous == :valid) {
                gaps << Hash(start => "#{date} #{hour}:00");
            }
            gaps[-1]{:count} := 0 ++;
            previous = :invalid;
        }
        ++hour;
    }
    say ("#{date}: #{ '%8s' % (valid ? ('%.3f' % Math.avg(valid...)) : 0) }",
         " mean from #{ '%2s' % valid.len } valid.");
}

var longest = gaps.sort_by{|a| -a{:count} }.first;

say ("Longest period of invalid readings was #{longest{:count}} hours,\n",
    "from #{longest{:start}} till #{longest{:end}}.");
```

Output:

```
1991-03-30:   10.000 mean from 24 valid.
1991-03-31:   23.542 mean from 24 valid.
1991-03-31:   40.000 mean from  1 valid.
1991-04-01:   23.217 mean from 23 valid.
1991-04-02:   19.792 mean from 24 valid.
1991-04-03:   13.958 mean from 24 valid.
Longest period of invalid readings was 24 hours,
from 1991-03-31 1:00 till 1991-04-01 1:00.
```

Output is from the sample of the task.

Text processing/2

```
var good_records = 0;
var dates = Hash.new;

ARGF.each { |line|
    var m = /^(\d\d\d\d-\d\d-\d\d)((?:\h+\d+\.\d+\h+-?\d+){24})\s*$/.match(line);
    m || (warn "Bad format at line #{$.}"; next);
    dates{m[0]} := 0 ++;
    var i = 0;
    m[1].words.all{|n| i++.is_even || (n.to_num >= 1) } && ++good_records;
}

say "#{good_records} good records out of #{$.} total";
say 'Repeated timestamps:';
say dates.to_a.grep{ .value > 1 }.map { .key }.sort.join("\n");
```

Output:

```
$ sidef script.sf < readings.txt
5017 good records out of 5471 total
Repeated timestamps:
1990-03-25
1991-03-31
1992-03-29
1993-03-28
1995-03-26
```

Text processing/Max licenses in use

```
var out = 0
var max_out = -1
var max_times = []

ARGF.each { |line|
    out += (line ~~ /OUT/ ? 1 : -1)
    if (out > max_out) {
        max_out = out
        max_times = []
    }
    if (out == max_out) {
        max_times << line.split(' ')[3]
    }
}

say "Maximum simultaneous license use is #{max_out} at the following times:"
max_times.each {|t| say "  #{t}" }
```

Output:

```
$ sidef max_licenses.sf < mlijobs.txt
Maximum simultaneous license use is 99 at the following times:
  2008/10/03_08:39:34
  2008/10/03_08:40:40
```

Textonyms

```
var words = ARGF.grep(/^[[:alpha:]]+\z/)

var dials = words.group_by {
    .tr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ',
        '22233344455566677778889999222333444555666777788899999')
}

var textonyms = dials.grep_v { .len > 1 }

say <<-"END"
    There are #{words.len} words which can be represented by the digit key mapping.
    They require #{dials.len} digit combinations to represent them.
    #{textonyms.len} digit combinations represent Textonyms.
    END

say "Top 5 in ambiguity:"
say textonyms.sort_by { |_,v| -v.len }.first(5).join("\n")

say "\nTop 5 in length:"
say textonyms.sort_by { |k,_| -k.len }.first(5).join("\n")
```

Output:

```
$ sidef textonyms.sf < unixdict.txt
There are 24978 words which can be represented by the digit key mapping.
They require 22903 digit combinations to represent them.
1473 digit combinations represent Textonyms.

Top 5 in ambiguity:
["729", ["paw", "pax", "pay", "paz", "raw", "ray", "saw", "sax", "say"]]
["269", ["amy", "any", "bmw", "bow", "box", "boy", "cow", "cox", "coy"]]
["2273", ["acre", "bard", "bare", "base", "cape", "card", "care", "case"]]
["726", ["pam", "pan", "ram", "ran", "sam", "san", "sao", "scm"]]
["782", ["pta", "pub", "puc", "pvc", "qua", "rub", "sub"]]

Top 5 in length:
["25287876746242", ["claustrophobia", "claustrophobic"]]
["7244967473642", ["schizophrenia", "schizophrenic"]]
["666628676342", ["onomatopoeia", "onomatopoeic"]]
["49376746242", ["hydrophobia", "hydrophobic"]]
["2668368466", ["contention", "convention"]]
```

The ISAAC cipher

```
require('Math::Random::ISAAC')

func xor_isaac(key, msg) {
  var rng = %O<Math::Random::ISAAC>.new(unpack('C*', key))

  msg.chars»ord()»                                          \
    -> »^« 256.of{ rng.irand % 95 + 32 }.last(msg.len).flip \
    -> «%« '%02X' -> join
}

var msg = 'a Top Secret secret'
var key = 'this is my secret key'

var enc = xor_isaac(key, msg)
var dec = xor_isaac(key, pack('H*', enc))

say "Message: #{msg}"
say "Key    : #{key}"
say "XOR    : #{enc}"
say "XOR dcr: #{pack('H*', dec)}"
```

Output:

```
Message: a Top Secret secret
Key    : this is my secret key
XOR    : 1C0636190B1260233B35125F1E1D0E2F4C5422
XOR dcr: a Top Secret secret
```

The Twelve Days of Christmas

```
var days = <first second third fourth fifth sixth seventh eighth ninth tenth eleventh twelfth>

var gifts = <<'EOT'.lines
  And a partridge in a pear tree.
  Two turtle doves,
  Three french hens,
  Four calling birds,
  Five golden rings,
  Six geese a-laying,
  Seven swans a-swimming,
  Eight maids a-milking,
  Nine ladies dancing,
  Ten lords a-leaping,
  Eleven pipers piping,
  Twelve drummers drumming,
EOT

func nth(n) { say "On the #{days[n]} day of Christmas, my true love gave to me:" }

nth(0)
say gifts[0].sub('And a', 'A')

for d (1..11) {
    say ''
    nth(d)
    for i flip(0..d) {
        say gifts[i]
    }
}
```

Output:

```
On the first day of Christmas, my true love gave to me:
  A partridge in a pear tree.

On the second day of Christmas, my true love gave to me:
  Two turtle doves,
  And a partridge in a pear tree.

On the third day of Christmas, my true love gave to me:
  Three french hens,
  Two turtle doves,
  And a partridge in a pear tree.

On the fourth day of Christmas, my true love gave to me:
  Four calling birds,
  Three french hens,
  Two turtle doves,
  And a partridge in a pear tree.
.
.
.
On the twelfth day of Christmas, my true love gave to me:
  Twelve drummers drumming,
  Eleven pipers piping,
  Ten lords a-leaping,
  Nine ladies dancing,
  Eight maids a-milking,
  Seven swans a-swimming,
  Six geese a-laying,
  Five golden rings,
  Four calling birds,
  Three french hens,
  Two turtle doves,
  And a partridge in a pear tree.
```

Thiele's interpolation formula

```
func thiele(x, y) {
    var ρ = {|i| [y[i]]*(y.len-i) }.map(^y)

    for i in ^(ρ.end) {
        ρ[i][1] = ((x[i] - x[i+1]) / (ρ[i][0] - ρ[i+1][0]))
    }
    for i (2 .. ρ.end) {
        for j (0 .. ρ.end-i) {
            ρ[j][i] = (((x[j]-x[j+i]) / (ρ[j][i-1]-ρ[j+1][i-1])) + ρ[j+1][i-2])
        }
    }

    var ρ0 = ρ[0]

    func t(xin) {
        var a = 0
        for i (ρ0.len ^.. 2) {
            a = ((xin - x[i-1]) / (ρ0[i] - ρ0[i-2] + a))
        }
        y[0] + ((xin-x[0]) / (ρ0[1]+a))
    }
    return t
}

# task 1: build 32 row trig table
var xVal = {|k| k * 0.05 }.map(^32)
var tSin = xVal.map { .sin }
var tCos = xVal.map { .cos }
var tTan = xVal.map { .tan }

# task 2: define inverses
var iSin = thiele(tSin, xVal)
var iCos = thiele(tCos, xVal)
var iTan = thiele(tTan, xVal)

# task 3: demonstrate identities
say 6*iSin(0.5)
say 3*iCos(0.5)
say 4*iTan(1)
```

Output:

```
3.14159265358979323846438729976818601771260734312
3.14159265358979323846157620314930763214337987744
3.14159265358979323846264318595256260456200366896
```

Thue-Morse

```
func recmap(repeat, seed, transform, callback) {
    func (repeat, seed) {
        callback(seed)
        repeat > 0 && __FUNC__(repeat-1, transform(seed))
    }(repeat, seed)
}

recmap(6, "0", {|s| s + s.tr('01', '10') }, { .say })
```

Output:

```
0
01
0110
01101001
0110100110010110
01101001100101101001011001101001
0110100110010110100101100110100110010110011010010110100110010110
```

Time a function

```
var benchmark = frequire('Benchmark')

func fac_rec(n) {
    n == 0 ? 1 : (n * __FUNC__(n - 1))
}

func fac_iter(n) {
    var prod = 1
    n.times { |i|
        prod *= i
    }
    prod
}

var result = benchmark.timethese(-3, Hash(
    'fac_rec'  => { fac_rec(20)  },
    'fac_iter' => { fac_iter(20) },
))

benchmark.cmpthese(result)
```

Output:

```
Benchmark: running fac_iter, fac_rec for at least 3 CPU seconds...
  fac_iter:  3 wallclock secs ( 3.23 usr +  0.00 sys =  3.23 CPU) @ 7331.89/s (n=23682)
   fac_rec:  3 wallclock secs ( 3.19 usr +  0.00 sys =  3.19 CPU) @ 3551.72/s (n=11330)
           Rate  fac_rec fac_iter
fac_rec  3552/s       --     -52%
fac_iter 7332/s     106%       --
```

Tokenize a string

```
'Hello,How,Are,You,Today'.split(',').join('.').say
```

Tokenize a string with escaping

```
func tokenize(string, sep, esc) {
    var fields = string.split(
        Regex(esc.escape + '.(*SKIP)(*FAIL)|' + sep.escape, 's'), -1
    )
    fields.map{.gsub(Regex(esc.escape + '(.)'), {|s1| s1 }) }
}

tokenize("one^|uno||three^^^^|four^^^|^cuatro|", '|', '^').each { |str|
    say str.dump
}
```

Output:

```
"one|uno"
""
"three^^"
"four^|cuatro"
""
```

Tonelli-Shanks algorithm

```
func tonelli(n, p) {
    legendre(n, p) == 1 || die "not a square (mod p)"
    var q = p-1
    var s = valuation(q, 2)
    s == 1 ? return(powmod(n, (p + 1) >> 2, p)) : (q >>= s)
    var c = powmod(2 ..^ p -> first {|z| legendre(z, p) == -1}, q, p)
    var r = powmod(n, (q + 1) >> 1, p)
    var t = powmod(n, q, p)
    var m = s
    var t2 = 0
    while (!p.divides(t - 1)) {
        t2 = ((t * t) % p)
        var b
        for i in (1 ..^ m) {
            if (p.divides(t2 - 1)) {
                b = powmod(c, 1 << (m - i - 1), p)
                m = i
                break
            }
            t2 = ((t2 * t2) % p)
        }

        r = ((r * b) % p)
        c = ((b * b) % p)
        t = ((t * c) % p)
    }
    return r
}

var tests = [
    [10, 13], [56, 101], [1030, 10009], [44402, 100049],
    [665820697, 1000000009], [881398088036, 1000000000039],
    [41660815127637347468140745042827704103445750172002, 10**50 + 577],
]

for n,p in tests {
    var r = tonelli(n, p)
    assert((r*r - n) % p == 0)
    say "Roots of #{n} are (#{r}, #{p-r}) mod #{p}"
}
```

Output:

```
Roots of 10 are (7, 6) mod 13
Roots of 56 are (37, 64) mod 101
Roots of 1030 are (1632, 8377) mod 10009
Roots of 44402 are (30468, 69581) mod 100049
Roots of 665820697 are (378633312, 621366697) mod 1000000009
Roots of 881398088036 are (791399408049, 208600591990) mod 1000000000039
Roots of 41660815127637347468140745042827704103445750172002 are (321029853699406208497419839873000389
```

Top rank per group

```
var data = <<'EOF'.lines.map{ <name id salary dept> ~Z .split(',') -> flat.to_h }
Tyler Bennett,E10297,32000,D101
John Rappl,E21437,47000,D050
George Woltman,E00127,53500,D101
Adam Smith,E63535,18000,D202
Claire Buckman,E39876,27800,D202
David McClellan,E04242,41500,D101
Rich Holcomb,E01234,49500,D202
Nathan Adams,E41298,21900,D050
Richard Potter,E43128,15900,D101
David Motsinger,E27002,19250,D202
Tim Sampair,E03033,27000,D101
Kim Arlich,E10001,57000,D190
Timothy Grove,E16398,29900,D190
EOF

var n = (ARGV ? Num(ARGV[0]) : "usage: #{__MAIN__} [n]\n".die)

for d in (data.map {|h| h{:dept} }.uniq.sort) {
    var es = data.grep { _{:dept} == d }.sort_by { -Num(_{:salary}) }
    say d
    n.times {
        es || break
        printf("%-15s | %-6s | %5d\n", es.shift(){<name id salary>...})
    }
    print "\n"
}
```

Output:

```
$ sidef top_rank.sf 3
D050
John Rappl      | E21437 | 47000
Nathan Adams    | E41298 | 21900

D101
George Woltman  | E00127 | 53500
David McClellan | E04242 | 41500
Tyler Bennett   | E10297 | 32000

D190
Kim Arlich      | E10001 | 57000
Timothy Grove   | E16398 | 29900

D202
Rich Holcomb    | E01234 | 49500
Claire Buckman  | E39876 | 27800
David Motsinger | E27002 | 19250
```

Topic variable

The underscore (_) topic variable is defined at compile-time in every block of a program. To call a method on it, we can just use the prefix dot (.) operator, followed by a method name, which is equivalent with *_.method*

```
say [9,16,25].map{ .sqrt }   # prints: [3, 4, 5]
```

Topological sort

```
func print_topo_sort(deps) {
    var ba = Hash()
    deps.each { |before, afters|
        afters.each { |after|
            if (before != after) {
                ba{before}{after} = 1
            }
            ba{after} \\= Hash()
        }
    }

    loop {
        var afters = ba.keys.grep {|k| ba{k}.values.len == 0 }.sort
        afters.len || break
        say afters.join(" ")
        ba.delete(afters...)
        ba.values.each { |v| v.delete(afters...) }
    }

    say (ba.len ? "Cicle found! #{ba.keys.sort.join(' ')}" : "---")
}

var deps = Hash(
    des_system_lib => < std synopsys std_cell_lib des_system_lib dw02
                                                  dw01 ramlib ieee >,
    dw01           => < ieee dw01 dware gtech                      >,
    dw02           => < ieee dw02 dware                            >,
    dw03           => < std synopsys dware dw03 dw02 dw01 ieee gtech  >,
    dw04           => < dw04 ieee dw01 dware gtech                 >,
    dw05           => < dw05 ieee dware                            >,
    dw06           => < dw06 ieee dware                            >,
    dw07           => < ieee dware                                 >,
    dware          => < ieee dware                                 >,
    gtech          => < ieee gtech                                 >,
    ramlib         => < std ieee                                   >,
    std_cell_lib   => < ieee std_cell_lib                          >,
    synopsys       => <                                            >,
)

print_topo_sort(deps)
deps{:dw01}.append('dw04')     # Add unresolvable dependency
print_topo_sort(deps)
```

Output:

```
ieee std synopsys
dware gtech ramlib std_cell_lib
dw01 dw02 dw05 dw06 dw07
des_system_lib dw03 dw04
---
ieee std synopsys
dware gtech ramlib std_cell_lib
dw02 dw05 dw06 dw07
Cicle found! des_system_lib dw01 dw03 dw04
```

Totient function

The Euler totient function is built-in as **Number.euler_phi()**, but we can easily re-implement it using its multiplicative property: **phi(p^k) = (p-1)*p^(k-1)**.

```
func φ(n) {
    n.factor_exp.prod {|p|
        (p[0]-1) * p[0]**(p[1]-1)
    }
}
```

```
for n in (1..25) {
    var totient = φ(n)
    printf("φ(%2s) = %3s%s\n", n, totient, totient==(n-1) ? ' - prime' : '')
}
```

```
[100, 1_000, 10_000, 100_000].each {|limit|
    var pi = (1..limit -> count_by {|n| φ(n) == (n-1) })
    say "Number of primes <= #{limit}: #{pi}"
}
```

Output:

```
Number of primes <= 100: 25
Number of primes <= 1000: 168
Number of primes <= 10000: 1229
Number of primes <= 100000: 9592
```

Towers of Hanoi

```
func hanoi(n, from=1, to=2, via=3) {
    if (n == 1) {
        say "Move disk from pole #{from} to pole #{to}."
    } else {
        hanoi(n-1, from, via,   to)
        hanoi(  1, from,  to,  via)
        hanoi(n-1,  via,  to, from)
    }
}

hanoi(4)
```

Trabb Pardo–Knuth algorithm

```
var nums; do {
    nums = Sys.readln("Please type 11 space-separated numbers: ").nums
} while(nums.len != 11)

nums.reverse.each { |n|
    var r = (n.abs.sqrt + (5 * n**3))
    say "#{n}\t#{ r > 400 ? 'Urk!' : r }"
}
```

Output:

```
Please type 11 space-separated numbers: 10 -1 1 2 3 4 4.3 4.305 4.303 4.302 4.301
4.301   399.886299747726800445468371077898575778355
4.302   Urk!
4.303   Urk!
4.305   Urk!
4.3     399.60864413533277208745589867998499263240 1
4       322
3       136.732050807568877293527446341505872366943
2       41.414213562373095048801688724209698078 57
1       6
-1      -4
10      Urk!
```

Tree traversal

```
func preorder(t) {
    t ? [t[0], __FUNC__(t[1])..., __FUNC__(t[2])...] : []
}

func inorder(t) {
    t ? [__FUNC__(t[1])..., t[0], __FUNC__(t[2])...] : []
}

func postorder(t) {
    t ? [__FUNC__(t[1])..., __FUNC__(t[2])..., t[0]] : []
}

func depth(t) {
    var a = [t]
    var ret = []
    while (a.len > 0) {
        var v = (a.shift \\ next)
        ret « v[0]
        a += [v[1,2]]
    }
    return ret
}

var x = [1,[2,[4,[7]],[5]],[3,[6,[8],[9]]]]
say "pre:   #{preorder(x)}"
say "in:    #{inorder(x)}"
say "post:  #{postorder(x)}"
say "depth: #{depth(x)}"
```

Output:

```
pre:   1 2 4 7 5 3 6 8 9
in:    7 4 2 5 1 8 6 9 3
post:  7 4 5 2 8 9 6 3 1
depth: 1 2 3 4 5 6 7 8 9
```

Trigonometric functions

```
var angle_deg = 45
var angle_rad = Num.pi/4

for arr in [
    [sin(angle_rad), sin(deg2rad(angle_deg))],
    [cos(angle_rad), cos(deg2rad(angle_deg))],
    [tan(angle_rad), tan(deg2rad(angle_deg))],
    [cot(angle_rad), cot(deg2rad(angle_deg))],
] {
    say arr.join(" ")
}

for n in [
    asin(sin(angle_rad)),
    acos(cos(angle_rad)),
    atan(tan(angle_rad)),
    acot(cot(angle_rad)),
] {
    say [n, rad2deg(n)].join(' ')
}
```

Output:

```
0.707106781186547 0.707106781186547
0.707106781186548 0.707106781186548
1 1
1 1
0.785398163397448 45
0.785398163397448 45
0.785398163397448 45
0.785398163397448 45
```

Triplet of three numbers

```
^6000 -> grep {|n| [-1, 3, 5].all {|k| n + k -> is_prime } }.say
```

Output:

```
[8, 14, 38, 68, 98, 104, 194, 224, 278, 308, 458, 614, 824, 854, 878, 1088, 1298, 1424, 1448, 1484, 166
```

Truncatable primes

```
func t_prime(n, left=true) {
    var p = %w(2 3 5 7);
    var f = (
        left ? { '1'..'9' ~X+ p }
             : { p ~X+ '1'..'9' }
    )
    n.times {
        p = f().grep{ .to_i.is_prime }
    }
    p.map{.to_i}.max
}

say t_prime(5, left: true)
say t_prime(5, left: false)
```

Output:

```
998443
739399
```

Truncate a file

```
func truncate(filename, len) {
    var file = File(filename)
    len > file.size ->
        && die "The provided length is greater than the length of the file"
    file.truncate(len)
}

# truncate "file.ext" to 1234 bytes
truncate("file.ext", 1234)
```

Truth table

A simple solution which accepts arbitrary user-input:

```
loop {
  var expr = Sys.readln("\nBoolean expression (e.g. 'a & b'): ").strip.lc
  break if expr.is_empty

  var vars = expr.scan(/[[:alpha:]]+/)
  if (vars.is_empty) {
    say "no variables detected in your boolean expression"
    next
  }

  var prefix = [];
  var suffix = [];

  vars.each { |v|
    print "#{v}\t"
    prefix << "[false, true].each { |#{v}|"
    suffix << "}"
  }
  say "| #{expr}"

  var body = ("say (" + vars.map{|v| v+",'\t'," }.join + " '| ', #{expr})")
  eval(prefix + [body] + suffix -> join("\n"))
}
```

Output:

```
Boolean expression (e.g. 'a & b'): (a & b) | c
a       b       c       | (a & b) | c
false   false   false   | false
false   false   true    | true
false   true    false   | false
false   true    true    | true
true    false   false   | false
true    false   true    | true
true    true    false   | true
true    true    true    | true
```

Twelve statements

```
var conditions = [
    { false },
    {|a| a.len == 13 },
    {|a| [a[7..12]].count(true) == 3 },
    {|a| [a[2..12 `by` 2]].count(true) == 2 },
    {|a| a[5] ? (a[6] && a[7]) : true },
    {|a| !a[2] && !a[3] && !a[4] },
    {|a| [a[1..11 `by` 2]].count(true) == 4 },
    {|a| a[2] == true^a[3] },
    {|a| a[7] ? (a[5] && a[6]) : true },
    {|a| [a[1..6]].count(true)  == 3 },
    {|a| [a[11,12]].count(true) == 2 },
    {|a| [a[7..9]].count(true)  == 1 },
    {|a| [a[1..11]].count(true) == 4 },
]

func miss(args) {
    1..12 -> grep {|i| conditions[i](args) != args[i] }
}

for k in (^(1<<12)) {
  var t  = ("0%012b" % k -> chars.map {|bit| bit == '1' })
  var no = miss(t)
  no.len == 0 && say "Solution: true statements are #{1..12->grep{t[_]}.join(' ')}"
  no.len == 1 && say "1 miss (#{no[0]}): true statements are #{1..12->grep{t[_]}.join(' ')}"
}
```

Output:

```
1 miss (1): true statements are 5 8 11
1 miss (1): true statements are 5 8 10 11 12
1 miss (1): true statements are 4 8 10 11 12
1 miss (8): true statements are 1 5
1 miss (11): true statements are 1 5 8
1 miss (12): true statements are 1 5 8 11
1 miss (12): true statements are 1 5 8 10 11 12
1 miss (8): true statements are 1 5 6 9 11
1 miss (8): true statements are 1 4
1 miss (12): true statements are 1 4 8 10 11 12
1 miss (6): true statements are 1 4 6 8 9
1 miss (7): true statements are 1 3 4 8 9
Solution: true statements are 1 3 4 6 7 11
1 miss (9): true statements are 1 3 4 6 7 9
1 miss (12): true statements are 1 2 4 7 9 12
1 miss (10): true statements are 1 2 4 7 9 10
1 miss (8): true statements are 1 2 4 7 8 9
```

Twin primes

```
func twin_primes_count(upto) {
    var count = 0
    var p1 = 2
    each_prime(3, upto, {|p2|
        if (p2 - p1 == 2) {
            ++count
        }
        p1 = p2
    })
    return count
}

for n in (1..9) {
    var count = twin_primes_count(10**n)
    say "There are #{count} twin primes <= 10^#{n}"
}
```

Output:

```
There are 2 twin primes <= 10^1
There are 8 twin primes <= 10^2
There are 35 twin primes <= 10^3
There are 205 twin primes <= 10^4
There are 1224 twin primes <= 10^5
There are 8169 twin primes <= 10^6
There are 58980 twin primes <= 10^7
There are 440312 twin primes <= 10^8
There are 3424506 twin primes <= 10^9
```

Two sum

```
func two_sum(numbers, sum) {
    var (i, j) = (0, numbers.end)
    while (i < j) {
        given (sum <=> numbers[i]+numbers[j]) {
            when (-1) { --j }
            when (+1) { ++i }
            default { return [i, j] }
        }
    }
    return []
}

say two_sum([0, 2, 11, 19, 90], 21)
say two_sum([0, 2, 11, 19, 90], 25)
```

Output:

```
[1, 3]
[]
```

Ulam numbers

```
func ulam(n) {

    static u     = Set(1,2)
    static ulams = [0, 1, 2]

    return ulams[n] if (ulams.end >= n)

    ++n

    for(var i = 3; true; ++i) {
        var count = 0

        ulams.each {|v|
            if (u.has(i - v) && (v != i-v)) {
                break if (count++ > 2)
            }
        }

        if (count == 2) {
            ulams << i
            u << i
            break if (ulams.len == n)
        }
    }

    ulams.tail
}

for k in (1..3) {
    say "The 10^#{k}-th Ulam number is: #{ulam(10**k)}"
}
```

Output:

```
The 10^1-th Ulam number is: 18
The 10^2-th Ulam number is: 690
The 10^3-th Ulam number is: 12294
```

Ulam spiral (for primes)

```
require('Imager')

var (n=512, start=1, file='ulam.png')

ARGV.getopt(
    'n=i' => \n,
    's=i' => \start,
    'f=s' => \file,
)

func cell(n, x, y, start) {
    y -= (n   >> 1)
    x -= (n-1 >> 1)
    var l = 2*(x.abs > y.abs ? x.abs : y.abs)
    var d = (y > x ? (l*3 + x + y) : (l - x - y))
    (l-1)**2 + d + start - 1
}

var black = %O<Imager::Color>.new('#000000')
var white = %O<Imager::Color>.new('#FFFFFF')

var img = %O<Imager>.new(xsize => n, ysize => n, channels => 1)
img.box(filled => 1, color => white)

for y=(^n), x=(^n) {
    if (cell(n, x, y, start).is_prime) {
        img.setpixel(x => x, y => y, color => black)
    }
}

img.write(file => file)
```

Output image

Ultra useful primes

```
say(" n   k")
say("----------")

for n in (1..13) {
    var t = 2**(2**n)
    printf("%2d   %d\n", n, {|k| t - k -> is_prob_prime }.first)
}
```

Output:

```
 n   k
----------
 1   1
 2   3
 3   5
 4   15
 5   5
 6   59
 7   159
 8   189
 9   569
10   105
11   1557
12   2549
13   2439
```

(takes ~20 seconds)

Unbias a random generator

```
func randN (n) {
    n.rand / (n-1) -> int
}

func unbiased(n) {
    var n1 = nil
    do { n1 = randN(n) } while (n1 == randN(n))
    return n1
}

var iterations = 1000

for n in (3..6) {
    var raw = []
    var fixed = []
    iterations.times {
          raw[    randN(n) ] := 0 ++
        fixed[ unbiased(n) ] := 0 ++
    }
    printf("N=%d   randN: %s, %4.1f%%   unbiased: %s, %4.1f%%\n",
        n, [raw, fixed].map {|a| (a.dump, a[1] * 100 / iterations) }...)
}
```

Output:

```
N=3   randN: [661, 339], 33.9%   unbiased: [497, 503], 50.3%
N=4   randN: [765, 235], 23.5%   unbiased: [493, 507], 50.7%
N=5   randN: [812, 188], 18.8%   unbiased: [509, 491], 49.1%
N=6   randN: [820, 180], 18.0%   unbiased: [510, 490], 49.0%
```

Undefined values

Sidef variables are initialized with a default *nil* value, representing the absence of a value.

```
var x      # declared, but not defined
x == nil   && say "nil value"
defined(x) || say "undefined"

# Give "x" some value
x = 42

defined(x) && say "defined"

# Change "x" back to `nil`
x = nil

defined(x) || say "undefined"
```

Output:

```
nil value
undefined
defined
undefined
```

Unicode strings

Sidef uses UTF-8 encoding for pretty much everything, including source files, strings, stdout, stderr and stdin.

```
# International class; name and street
 class 国際( なまえ, Straße ) {

    # Say who am I!
    method 言え {
        say "I am #{self.なまえ} from #{self.Straße}"
    }
}

 # all the people of the world!
 var 民族 = [
              国際( "高田  Friederich", "台湾" ),
              国際( "Smith Σωκράτης", "Cantù" ),
              国際( "Stanisław Lec", "południow" ),
           ]

民族.each { |garçon|
    garçon.言え
}
```

Output:

```
I am 高田  Friederich from 台湾
I am Smith Σωκράτης from Cantù
I am Stanisław Lec from południow
```

Unicode variable names

```
var Δ = 1
Δ += 1
say Δ
```

Output:

```
2
```

Universal Turing machine

```
func run_utm(state="", blank="", rules=[], tape=[blank], halt="", pos=0) {

    if (pos < 0) {
        pos += tape.len;
    }

    if (pos !~ tape.range) {
        die "Bad initial position";
    }

    loop {
        print "#{state}\t";
        tape.range.each { |i|
            var v = tape[i];
            print (i == pos ? "[#{v}]" : " #{v} ");
        };
        print "\n";

        if (state == halt) {
            break;
        }

        for s0, v0, v1, dir, s1 in rules {
            if ((s0 != state) || (tape[pos] != v0)) {
                next;
            }

            tape[pos] = v1;

            given(dir) {
                when ('left') {
                     if (pos == 0) { tape.unshift(blank) }
                     else          { --pos };
                }
                when ('right') {
                    if (++pos >= tape.len) {
                        tape.append(blank)
                    }
                }
            }

            state = s1;
            goto :NEXT;
        }

        die 'No matching rules';
        @:NEXT;
    }
}
```

```
print "incr machine\n";
run_utm(
    halt:  'qf',
    state: 'q0',
    tape:  %w(1 1 1),
    blank: 'B',
    rules: [
        %w(q0 1 1 right q0),
        %w(q0 B 1 stay  qf),
    ]);

say "\nbusy beaver";
run_utm(
    halt:  'halt',
    state: 'a',
    blank: '0',
    rules: [
        %w(a 0 1 right b),
        %w(a 1 1 left  c),
        %w(b 0 1 left  a),
        %w(b 1 1 right b),
        %w(c 0 1 left  b),
        %w(c 1 1 stay  halt),
    ]);

say "\nsorting test";
run_utm(
    halt:  'STOP',
    state: 'A',
    blank: '0',
    tape:  %w(2 2 2 1 2 2 1 2 1 2 1 2 1 2),
    rules: [
        %w(A 1 1 right A),
        %w(A 2 3 right B),
        %w(A 0 0 left  E),
        %w(B 1 1 right B),
        %w(B 2 2 right B),
        %w(B 0 0 left  C),
        %w(C 1 2 left  D),
        %w(C 2 2 left  C),
        %w(C 3 2 left  E),
        %w(D 1 1 left  D),
        %w(D 2 2 left  D),
        %w(D 3 1 right A),
        %w(E 1 1 left  E),
        %w(E 0 0 right STOP),
    ]);
```

Unix/ls

Explicit, by opening the current working directory:

```
var content = []
Dir.cwd.open.each { |file|
    file ~~ < . .. > && next
    content.append(file)
}

content.sort.each { |file|
    say file
}
```

Implicit, by using the *String.glob* method:

```
'*'.glob.each { |file|
    say file
}
```

Unprimeable numbers

```
func is_unprimeable(n) {
    var t = 10*floor(n/10)
    for k in (t+1 .. t+9 `by` 2) {
        return false if k.is_prime
    }

    if (n.is_div(2) || n.is_div(5)) {
        return true if !is_prime(n%10)
        return true if (n % 10**n.ilog(10) > 9)
    }

    for k in (1 .. n.ilog(10)) {
        var u = 10**k
        var v = (n - (u * (floor(n/u) % 10)))
        0..9 -> any {|d| is_prime(v + d*u) } && return false
    }

    return true
}

with (35) {|n|
    say ("First #{n} unprimeables:\n", is_unprimeable.first(n).join(' '))
}

with (600) {|n|
    say ("\n#{n}th unprimeable: ", is_unprimeable.nth(n), "\n")
}

for d in (0..9) {
    say ("First unprimeable that ends with #{d}: ",
        1..Inf -> lazy.map {|k| k*10 + d }.grep(is_unprimeable).first)
}
```

Output:

```
First 35 unprimeables:
200 204 206 208 320 322 324 325 326 328 510 512 514 515 516 518 530 532 534 535 536 538 620 622 624 625

600th unprimeable: 5242

First unprimeable that ends with 0: 200
First unprimeable that ends with 1: 595631
First unprimeable that ends with 2: 322
First unprimeable that ends with 3: 1203623
First unprimeable that ends with 4: 204
First unprimeable that ends with 5: 325
First unprimeable that ends with 6: 206
First unprimeable that ends with 7: 872897
First unprimeable that ends with 8: 208
First unprimeable that ends with 9: 212159
```

URL decoding

```
func urldecode(str) {
    str.gsub!('+', ' ')
    str.gsub!(/\%([A-Fa-f0-9]{2})/, {|a| 'C'.pack(a.hex) })
    return str
}

say urldecode('http%3A%2F%2Ffoo+bar%2F')   #=> "http://foo bar/"
```

URL encoding

```
func urlencode(str) {
    str.gsub!(%r"([^-A-Za-z0-9_.!~*'() ])", {|a| "%%%02X" % a.ord })
    str.gsub!(' ', '+')
    return str
}

say urlencode('http://foo bar/')
```

Output:

```
http%3A%2F%2Ffoo+bar%2F
```

User input/Graphical

```
var gtk2 = require('Gtk2') -> init

var gui = %O<Gtk2::Builder>.new
gui.add_from_string(DATA.slurp)

func clicked_ok(*_) {
    var entry = gui.get_object('entry1')
    var text = entry.get_text
```

```
    var spinner = gui.get_object('spinbutton1')
    var number = spinner.get_text

    say "string: #{text}"
    say "number: #{number}"

    number == 75000 ? gtk2.main_quit : warn "Invalid number!"
}

func clicked_cancel(*_) {
    gtk2.main_quit
}

gui.get_object('button1').signal_connect('clicked', clicked_ok)
gui.get_object('button2').signal_connect('clicked', clicked_cancel)

gtk2.main

__DATA__
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <requires lib="gtk+" version="2.24"/>
  <!-- interface-naming-policy project-wide -->
  <object class="GtkAdjustment" id="adjustment1">
    <property name="upper">100000</property>
    <property name="value">75000</property>
    <property name="step_increment">1</property>
    <property name="page_increment">10</property>
  </object>
  <object class="GtkWindow" id="window1">
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <child>
      <object class="GtkVBox" id="vbox1">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <child>
          <object class="GtkLabel" id="label1">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="label" translatable="yes">Please insert a string and a number:</property>
          </object>
          <packing>
            <property name="expand">True</property>
            <property name="fill">True</property>
            <property name="position">0</property>
          </packing>
        </child>
        <child>
          <object class="GtkHBox" id="hbox1">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <child>
              <object class="GtkLabel" id="label2">
                <property name="visible">True</property>
                <property name="can_focus">False</property>
                <property name="label" translatable="yes">string:</property>
              </object>
              <packing>
                <property name="expand">True</property>
                <property name="fill">True</property>
                <property name="padding">55</property>
                <property name="position">0</property>
              </packing>
            </child>
```

```
      <child>
        <object class="GtkEntry" id="entry1">
          <property name="visible">True</property>
          <property name="can_focus">True</property>
          <property name="invisible_char">•</property>
          <property name="primary_icon_activatable">False</property>
          <property name="secondary_icon_activatable">False</property>
          <property name="primary_icon_sensitive">True</property>
          <property name="secondary_icon_sensitive">True</property>
        </object>
        <packing>
          <property name="expand">True</property>
          <property name="fill">True</property>
          <property name="position">1</property>
        </packing>
      </child>
    </object>
    <packing>
      <property name="expand">True</property>
      <property name="fill">True</property>
      <property name="position">1</property>
    </packing>
  </child>
  <child>
    <object class="GtkHBox" id="hbox2">
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <child>
        <object class="GtkLabel" id="label3">
          <property name="visible">True</property>
          <property name="can_focus">False</property>
          <property name="label" translatable="yes">number:</property>
        </object>
        <packing>
          <property name="expand">True</property>
          <property name="fill">True</property>
          <property name="position">0</property>
        </packing>
      </child>
      <child>
        <object class="GtkSpinButton" id="spinbutton1">
          <property name="visible">True</property>
          <property name="can_focus">True</property>
          <property name="invisible_char">•</property>
          <property name="primary_icon_activatable">False</property>
          <property name="secondary_icon_activatable">False</property>
          <property name="primary_icon_sensitive">True</property>
          <property name="secondary_icon_sensitive">True</property>
          <property name="adjustment">adjustment1</property>
          <property name="numeric">True</property>
        </object>
        <packing>
          <property name="expand">True</property>
          <property name="fill">True</property>
          <property name="position">1</property>
        </packing>
      </child>
    </object>
    <packing>
      <property name="expand">True</property>
      <property name="fill">True</property>
      <property name="position">2</property>
    </packing>
  </child>
  <child>
    <object class="GtkHButtonBox" id="hbuttonbox1">
```

```
            <object class="GtkButtonBox" id="hbuttonbox1">
              <property name="visible">True</property>
              <property name="can_focus">False</property>
              <property name="layout_style">spread</property>
              <child>
                <object class="GtkButton" id="button1">
                  <property name="label">gtk-ok</property>
                  <property name="visible">True</property>
                  <property name="can_focus">True</property>
                  <property name="receives_default">True</property>
                  <property name="use_stock">True</property>
                </object>
                <packing>
                  <property name="expand">False</property>
                  <property name="fill">False</property>
                  <property name="position">0</property>
                </packing>
              </child>
              <child>
                <object class="GtkButton" id="button2">
                  <property name="label">gtk-cancel</property>
                  <property name="visible">True</property>
                  <property name="can_focus">True</property>
                  <property name="receives_default">True</property>
                  <property name="use_stock">True</property>
                </object>
                <packing>
                  <property name="expand">False</property>
                  <property name="fill">False</property>
                  <property name="position">1</property>
                </packing>
              </child>
            </object>
            <packing>
              <property name="expand">True</property>
              <property name="fill">True</property>
              <property name="position">3</property>
            </packing>
          </child>
        </object>
      </child>
    </object>
</interface>
```

User input/Text

Using the *read(Type)* built-in function:

```
var s = read(String)
var i = read(Number)     # auto-conversion to a number
```

or using the *Sys.readln(msg)* method:

```
var s = Sys.readln("Enter a string: ")
var i = Sys.readln("Enter a number: ").to_i
```

UTF-8 encode and decode

```
func utf8_encoder(Number code) {
    code.chr.encode('UTF-8').bytes.map{.chr}
}

func utf8_decoder(Array bytes) {
    bytes.map{.ord}.decode('UTF-8')
}

for n in ([0x0041, 0x00F6, 0x0416, 0x20AC, 0x1D11E]) {
    var encoded = utf8_encoder(n)
    var decoded = utf8_decoder(encoded)
    assert_eq(n, decoded.ord)
    say "#{decoded} -> #{encoded}"
}
```

Output:

```
A -> ["A"]
ö -> ["\xC3", "\xB6"]
Ж -> ["\xD0", "\x96"]
€ -> ["\xE2", "\x82", "\xAC"]
𝄞 -> ["\xF0", "\x9D", "\x84", "\x9E"]
```

Vampire number

```
func is_vampire (n) {
    return [] if n.ilog10.is_even

    var l1 = n.isqrt.ilog10.ipow10
    var l2 = n.isqrt

    var s = n.digits.sort.join

    gather {
        n.divisors.each { |d|

            d < l1 && next
            d > l2 && break

            var t = n/d

            next if (d%%10 && t%%10)
            next if ("#{d}#{t}".sort != s)

            take([d, t])
        }
    }
}

say "First 25 Vampire Numbers:"

with (1) { |i|
    for (var n = 1; i <= 25; ++n) {
        var fangs = is_vampire(n)
        printf("%2d. %6s : %s\n", i++, n, fangs.join(' ')) if fangs
    }
}

say "\nIndividual tests:"

[16758243290880, 24959017348650, 14593825548650].each { |n|
    var fangs = is_vampire(n)
    say "#{n}: #{fangs ? fangs.join(', ') : 'is not a vampire number'}"
}
```

Output:

```
First 25 Vampire Numbers:
 1.   1260 : [21, 60]
 2.   1395 : [15, 93]
 3.   1435 : [35, 41]
 4.   1530 : [30, 51]
 5.   1827 : [21, 87]
 6.   2187 : [27, 81]
 7.   6880 : [80, 86]
 8. 102510 : [201, 510]
 9. 104260 : [260, 401]
10. 105210 : [210, 501]
11. 105264 : [204, 516]
12. 105750 : [150, 705]
13. 108135 : [135, 801]
14. 110758 : [158, 701]
15. 115672 : [152, 761]
16. 116725 : [161, 725]
17. 117067 : [167, 701]
18. 118440 : [141, 840]
19. 120600 : [201, 600]
20. 123354 : [231, 534]
21. 124483 : [281, 443]
22. 125248 : [152, 824]
23. 125433 : [231, 543]
24. 125460 : [204, 615] [246, 510]
25. 125500 : [251, 500]

Individual tests:
16758243290880: [1982736, 8452080], [2123856, 7890480], [2751840, 6089832], [2817360, 5948208]
24959017348650: [2947050, 8469153], [2949705, 8461530], [4125870, 6049395], [4129587, 6043950], [423076
14593825548650: is not a vampire number
```

Van der Corput sequence

```
func vdc(value, base=2) {
    while (value[-1] > 0) {
        value.append(value[-1] / base -> int)
    }
    var (x, sum) = (1, 0)
    value.each { |i|
        sum += ((i % base) / (x *= base))
    }
    return sum
}

for base in (2..5) {
    var seq = 10.of {|i| vdc([i], base) }
    "base %d: %s\n".printf(base, seq.map{|n| "%.4f" % n}.join(', '))
}
```

Output:

```
base 2: 0.0000, 0.5000, 0.2500, 0.7500, 0.1250, 0.6250, 0.3750, 0.8750, 0.0625, 0.5625
base 3: 0.0000, 0.3333, 0.6667, 0.1111, 0.4444, 0.7778, 0.2222, 0.5556, 0.8889, 0.0370
base 4: 0.0000, 0.2500, 0.5000, 0.7500, 0.0625, 0.3125, 0.5625, 0.8125, 0.1250, 0.3750
base 5: 0.0000, 0.2000, 0.4000, 0.6000, 0.8000, 0.0400, 0.2400, 0.4400, 0.6400, 0.8400
```

Van Eck sequence

```
func van_eck(n) {

    var seen = Hash()
    var seq  = [0]
    var prev = seq[-1]

    for k in (1 ..^ n) {
        seq << (seen.has(prev) ? (k - seen{prev}) : 0)
        seen{prev} = k
        prev = seq[-1]
    }

    seq
}

say van_eck(10)
say van_eck(1000).slice(991-1, 1000-1)
```

Output:

```
[0, 0, 1, 0, 2, 0, 2, 2, 1, 6]
[4, 7, 30, 25, 67, 225, 488, 0, 10, 136]
```

Variable-length quantity

```
func vlq_encode(num) {
    var t = (0x7F & num)
    var str = t.chr
    while (num >>= 7) {
       t = (0x7F & num)
       str += chr(0x80 | t)
    }
    str.reverse
}

func vlq_decode(str) {
    var num = ''
    str.each_byte { |b|
        num += ('%07b' % (b & 0x7F))
    }
    Num(num, 2)
}

var tests = [0,   0xa,   123,   254,   255,   256,
             257, 65534, 65535, 65536, 65537, 0x1fffff,
             0x200000]

tests.each { |t|
    var vlq = vlq_encode(t)
    printf("%8s %12s %8s\n", t,
        vlq.bytes.join(':', { "%02X" % _ }), vlq_decode(vlq))
}
```

Output:

```
       0           00        0
      10           0A       10
     123           7B      123
     254        81:7E      254
     255        81:7F      255
     256        82:00      256
     257        82:01      257
   65534     83:FF:7E    65534
   65535     83:FF:7F    65535
   65536     84:80:00    65536
   65537     84:80:01    65537
 2097151     FF:FF:7F  2097151
 2097152  81:80:80:00  2097152
```

Variadic function

A parameter declared with "*", can take any number of arguments of any type.

```
func print_all(*things) {
    things.each { |x| say x }
}
```

This function can be called with any number of arguments:

```
print_all(4, 3, 5, 6, 4, 3)
print_all(4, 3, 5)
print_all("Rosetta", "Code", "Is", "Awesome!")
```

Also, there is "..." which transforms an array into a list of arguments.

```
var args = ["Rosetta", "Code", "Is", "Awesome!"]
print_all(args...)
```

Vector

```
class MyVector(:args) {

    has Number x
    has Number y

    method init {
        if ([:x, :y] ~~ args) {
            x = args{:x}
            y = args{:y}
        }
        elsif ([:length, :angle] ~~ args) {
            x = args{:length}*args{:angle}.cos
            y = args{:length}*args{:angle}.sin
        }
        elsif ([:from, :to] ~~ args) {
            x = args{:to}[0]-args{:from}[0]
            y = args{:to}[1]-args{:from}[1]
        }
        else {
            die "Invalid arguments: #{args}"
        }
    }

    method length { hypot(x, y) }
    method angle  { atan2(y, x) }

    method +(MyVector v) { MyVector(x => x + v.x,  y => y + v.y) }
    method -(MyVector v) { MyVector(x => x - v.x,  y => y - v.y) }
    method *(Number n)   { MyVector(x => x * n,    y => y * n)   }
    method /(Number n)   { MyVector(x => x / n,    y => y / n)   }

    method neg  { self * -1 }
    method to_s { "vec[#{x}, #{y}]" }
}

var u = MyVector(x => 3, y => 4)
var v = MyVector(from => [1, 0], to => [2, 3])
var w = MyVector(length => 1, angle => 45.deg2rad)

say u    #: vec[3, 4]
say v    #: vec[1, 3]
say w    #: vec[0.70710678118654752440084436210485, 0.70710678118654752440084436210485]

say u.length                            #: 5
say u.angle.rad2deg                     #: 53.130102354155978703144387440906 59

say u+v                                 #: vec[4, 7]
say u-v                                 #: vec[2, 1]
say -u                                  #: vec[-3, -4]
say u*10                                #: vec[30, 40]
say u/2                                 #: vec[1.5, 2]
```

Vector products

```
class MyVector(x, y, z) {
    method ·(vec) {
        [self{:x,:y,:z}] »*« [vec{:x,:y,:z}] «+»
    }

    method ×(vec) {
        MyVector(self.y*vec.z - self.z*vec.y,
               self.z*vec.x - self.x*vec.z,
               self.x*vec.y - self.y*vec.x)
    }

    method to_s {
        "(#{x}, #{y}, #{z})"
    }
}

var a = MyVector(3, 4, 5)
var b = MyVector(4, 3, 5)
var c = MyVector(-5, -12, -13)

say "a=#{a}; b=#{b}; c=#{c};"
say "a · b = #{a · b}"
say "a × b = #{a × b}"
say "a · (b × c) = #{a · (b × c)}"
say "a × (b × c) = #{a × (b × c)}"
```

Output:

```
a=(3, 4, 5); b=(4, 3, 5); c=(-5, -12, -13);
a · b = 49
a × b = (5, 5, -7)
a · (b × c) = 6
a × (b × c) = (-267, 204, -3)
```

Verify distribution uniformity/Chi-squared test

```
# Confluent hypergeometric function of the first kind F_1(a;b;z)
func F1(a, b, z, limit=100) {
    sum(0..limit, {|k|
        rising_factorial(a, k) / rising_factorial(b, k) * z**k / k!
    })
}

func γ(a,x) {    # lower incomplete gamma function γ(a,x)
    #a**(-1) * x**a * F1(a, a+1, -x)           # simpler formula
    a**(-1) * x**a * exp(-x) * F1(1, a+1, x)    # slightly better convergence
}

func P(a,z) {    # regularized gamma function P(a,z)
    γ(a,z) / Γ(a)
}

func chi_squared_cdf (k, x) {
    var f = (k<20 ? 20 : 10)
    given(x) {
        when (0) { 0 }
        case (. < (k + f*sqrt(k))) { P(k/2, x/2) }
        else { 1 }
    }
}

func chi_squared_test(arr, significance = 0.05) {
    var n = arr.len
    var N = arr.sum
    var expected = N/n
    var χ_squared = arr.sum_by {|v| (v-expected)**2 / expected }
    var p_value = (1 - chi_squared_cdf(n-1, χ_squared))
    [χ_squared, p_value, p_value > significance]
}

[
    %n< 199809 200665 199607 200270 199649 >,
    %n< 522573 244456 139979  71531  21461 >,
].each {|dataset|
    var r = chi_squared_test(dataset)
    say "data: #{dataset}"
    say "χ² = #{r[0]}, p-value = #{r[1].round(-4)}, uniform = #{r[2]}\n"
}
```

Output:

```
data: [199809, 200665, 199607, 200270, 199649]
χ² = 4.14628, p-value = 0.3866, uniform = true

data: [522573, 244456, 139979, 71531, 21461]
χ² = 790063.27594, p-value = 0, uniform = false
```

Vigenère cipher

```
func s2v(s) { s.uc.scan(/[A-Z]/).map{.ord} »-» 65 }
func v2s(v) { v »%» 26 »+» 65 -> map{.chr}.join   }

func blacken (red, key) { v2s(s2v(red) »+« s2v(key)) }
func redden  (blk, key) { v2s(s2v(blk) »-« s2v(key)) }

var red = "Beware the Jabberwock, my son! The jaws that bite, the claws that catch!"
var key = "Vigenere Cipher!!!"

say red
say (var black = blacken(red, key))
say redden(black, key)
```

Output:

```
Beware the Jabberwock, my son! The jaws that bite, the claws that catch!
WMCEEIKLGRPIFVMEUGXQPWQVIOIAVEYXUEKFKBTALVXTGAFXYEVKPAGY
BEWARETHEJABBERWOCKMYSONTHEJAWSTHATBITETHECLAWSTHATCATCH
```

Visualize a tree

```
func visualize_tree(tree, label, children,
                    indent = '',
                    mids = ['├─', '| '],
                    ends = ['└─', '  '],
) {
    func visit(node, pre) {
        gather {
            take(pre[0] + label(node))
            var chldn = children(node)
            var end = chldn.end
            chldn.each_kv { |i, child|
                if (i == end) { take(visit(child, [pre[1]] ~X+ ends)) }
                else          { take(visit(child, [pre[1]] ~X+ mids)) }
            }
        }
    }
    visit(tree, [indent] * 2)
}

var tree = 'root':['a':['a1':['a11':[]]],'b':['b1':['b11':[]],'b2':[],'b3':[]]]
say visualize_tree(tree, { .first }, { .second }).flatten.join("\n")
```

Output:

```
root
├─a
│ └─a1
│   └─a11
└─b
  ├─b1
  │ └─b11
  ├─b2
  └─b3
```

Vogel's approximation method

```
var costs = :(
    W => :(A => 16, B => 16, C => 13, D => 22, E => 17),
    X => :(A => 14, B => 14, C => 13, D => 19, E => 15),
    Y => :(A => 19, B => 19, C => 20, D => 23, E => 50),
    Z => :(A => 50, B => 12, C => 50, D => 15, E => 11)
)

var demand = :(A => 30, B => 20, C => 70, D => 30, E => 60)
var supply = :(W => 50, X => 60, Y => 50, Z => 50)

var cols = demand.keys.sort
var res = :(); costs.each {|k| res{k} = :() }
var g   = :(); supply.each {|x| g{x} = costs{x}.keys.sort_by{|g| costs{x}{g} }}

demand.each {|x| g{x} = costs.keys.sort_by{|g| costs{g}{x} }}

while (g) {
    var d = demand.collect {|x|
        [x, var z = costs{g{x}[0]}{x}, g{x}[1] ? costs{g{x}[1]}{x}-z : z]
    }

    var s = supply.collect {|x|
        [x, var z = costs{x}{g{x}[0]}, g{x}[1] ? costs{x}{g{x}[1]}-z : z]
    }

    with (d.max_by { .[2] }) { |dmax|
        d.grep! { .[2] == dmax[2] }.min_by! { .[1] }
    }

    with (s.max_by { .[2] }) { |dmax|
        s.grep! { .[2] == dmax[2] }.min_by! { .[1] }
    }

    var (t,f) = (d[2] == s[2] ? ((s[1], d[1])) : ((d[2], s[2])))
        (d,s) = (t > f ? ((d[0], g{d[0]}[0])) : ((g{s[0]}[0],s[0])))

    var v = (supply{s} `min` demand{d})

    res{s}{d} := 0 += v
    demand{d} -= v

    if (demand{d} == 0) {
        supply.grep {|_,n| n != 0 }.each {|x| g{x}.delete(d) }
        g.delete(d)
        demand.delete(d)
    }

    supply{s} -= v

    if (supply{s} == 0) {
        demand.grep {|_,n| n != 0 }.each {|x| g{x}.delete(s) }
        g.delete(s)
        supply.delete(s)
    }
}

say("\t", cols.join("\t"))

var cost = 0
costs.keys.sort.each { |g|
  print(g, "\t")
```

```
  cols.each { |n|
    if (defined(var y = res{g}{n})) {
        print(y)
        cost += (y * costs{g}{n})
    }
    print("\t")
  }
  print("\n")
}

say "\n\nTotal Cost = #{cost}"
```

Output:

```
        A       B       C       D       E
W                       50
X       30              20              10
Y               20              30
Z                                       50


Total Cost = 3100
```

Voronoi diagram

```
require('Imager')

func generate_voronoi_diagram(width, height, num_cells) {
    var img = %O<Imager>.new(xsize => width, ysize => height)
    var (nx,ny,nr,ng,nb) = 5.of { [] }...

    for i in (^num_cells) {
        nx << rand(^width)
        ny << rand(^height)
        nr << rand(^256)
        ng << rand(^256)
        nb << rand(^256)
    }

    for y=(^height), x=(^width) {
        var j = (^num_cells -> min_by {|i| hypot(nx[i]-x, ny[i]-y) })
        img.setpixel(x => x, y => y, color => [nr[j], ng[j], nb[j]])
    }
    return img
}

var img = generate_voronoi_diagram(500, 500, 25)
img.write(file => 'VoronoiDiagram.png')
```

Output image

Walk a directory/Non-recursively

```
'*.p[lm]'.glob.each { |file| say file }    # Perl files under this directory
```

Output:

```
x.pl
x.pm
```

```
func file_match(Block callback, pattern=/\.txt\z/, path=Dir.cwd) {
    path.open(\var dir_h) || return nil
    dir_h.entries.each { |entry|
        if (entry.basename ~~ pattern) {
            callback(entry)
        }
    }
}

file_match(
    path: %d'/tmp',
    pattern: /\.p[lm]\z/i,
    callback: { |file|
        say file;
    }
)
```

Output:

```
/tmp/x.pl
/tmp/x.pm
```

Walk a directory/Recursively

```
func traverse(Block callback, Dir dir) {
    dir.open(\var dir_h) || return nil

    dir_h.entries.each { |entry|
        if (entry.is_a(Dir)) {
            traverse(callback, entry)
        } else {
            callback(entry)
        }
    }
}

var dir = Dir.cwd
var pattern = /foo/   # display files that contain 'foo'

traverse(
    { |file|
        if (file.basename ~~ pattern) {
            say file
        }
    } => dir
)
```

Water collected between towers

```
func max_l(Array a, m = a[0]) {
    gather { a.each {|e| take(m = max(m, e)) } }
}

func max_r(Array a) {
    max_l(a.flip).flip
}

func water_collected(Array towers) {
    var levels = (max_l(towers) »min« max_r(towers))
    (levels »-« towers).grep{ _ > 0 }.sum
}

[
    [ 1, 5,  3, 7, 2 ],
    [ 5, 3,  7, 2, 6, 4, 5, 9, 1, 2 ],
    [ 2, 6,  3, 5, 2, 8, 1, 4, 2, 2, 5, 3, 5, 7, 4, 1 ],
    [ 5, 5,  5, 5 ],
    [ 5, 6,  7, 8 ],
    [ 8, 7,  7, 6 ],
    [ 6, 7, 10, 7, 6 ],
].map { water_collected(_) }.say
```

Output:

```
[2, 14, 35, 0, 0, 0, 0]
```

Web scraping

```
var ua = frequire('LWP::Simple')
var url = 'http://tycho.usno.navy.mil/cgi-bin/timer.pl'
var match = /<BR>(.+? UTC)/.match(ua.get(url))
say match[0] if match
```

Output:

```
Oct. 27, 00:20:50 UTC
```

Weird numbers

```
func is_pseudoperfect(n, d = n.divisors.slice(0, -2), s = d.sum, m = d.end) {

    return false if (m < 0)

    while (d[m] > n) {
        s -= d[m--]
    }

    return true if (n == s)
    return true if (d[m] == n)

    __FUNC__(n-d[m], d, s-d[m], m-1) || __FUNC__(n, d, s-d[m], m-1)
}

func is_weird(n) {
    (n.sigma > 2*n) && !is_pseudoperfect(n)
}

var w = (1..Inf -> lazy.grep(is_weird).first(25))
say "The first 25 weird numbers:\n#{w.join(' ')}"
```

Output:

```
The first 25 weird numbers:
70 836 4030 5830 7192 7912 9272 10430 10570 10792 10990 11410 11690 12110 12530 12670 13370 13510 13790
```

Welch's t-test

```
func p_value (A, B) {
    [A.len, B.len].all { _ > 1 } || return 1

    var x̄_a = Math.avg(A...)
    var x̄_b = Math.avg(B...)

    var a_var = (A.map {|n| (x̄_a - n)**2 }.sum / A.end)
    var b_var = (B.map {|n| (x̄_b - n)**2 }.sum / B.end)

    (a_var && b_var) || return 1

    var Welsh_t_statistic = ((x̄_a - x̄_b) / √(a_var/A.len + b_var/B.len))

    var DoF = ((a_var/A.len + b_var/B.len)**2 /
              ((a_var**2 / (A.len**3 - A.len**2)) + (b_var**2 / (B.len**3 - B.len**2))))

    var sa = (DoF/2 - 1)
    var x  = (DoF/(Welsh_t_statistic**2 + DoF))
    var N  = 65355
    var h  = x/N

    var (sum1=0, sum2=0)

    ^N -> lazy.map { _ * h }.each { |i|
        sum1 += (((i + h/2) ** sa) / √(1 - (i + h/2)))
        sum2 += ((  i       ** sa) / √(1 - (i    )))
    }

    (h/6 * (x**sa / √(1-x) + 4*sum1 + 2*sum2)) /
        (gamma(sa + 1) * √(Num.pi) / gamma(sa + 1.5))
}

# Testing
var tests = [
  %n<27.5 21.0 19.0 23.6 17.0 17.9 16.9 20.1 21.9 22.6 23.1 19.6 19.0 21.7 21.4>,
  %n<27.1 22.0 20.8 23.4 23.4 23.5 25.8 22.0 24.8 20.2 21.9 22.1 22.9 20.5 24.4>,

  %n<17.2 20.9 22.6 18.1 21.7 21.4 23.5 24.2 14.7 21.8>,
  %n<21.5 22.8 21.0 23.0 21.6 23.6 22.5 20.7 23.4 21.8 20.7 21.7 21.5 22.5 23.6 21.5 22.5 23.5 21.5
21.8>,

  %n<19.8 20.4 19.6 17.8 18.5 18.9 18.3 18.9 19.5 22.0>,
  %n<28.2 26.6 20.1 23.3 25.2 22.1 17.7 27.6 20.6 13.7 23.2 17.5 20.6 18.0 23.9 21.6 24.3 20.4 24.0
13.2>,

  %n<30.02 29.99 30.11 29.97 30.01 29.99>,
  %n<29.89 29.93 29.72 29.98 30.02 29.98>,

  %n<3.0 4.0 1.0 2.1>,
  %n<490.2 340.0 433.9>
]

tests.each_slice(2, {|left, right|
    say p_value(left, right)
})
```

Output:

```
0.021378001462867032506111328138722020511151931775 6
0.148841696605327985083613019511085971435711697961
0.035972271029796718087136761853897744693324815065 1
0.090773324285666887884095627552353608340669252565 6
0.010751534033392975546532371802885666993291203101 2
```

Wieferich primes

```
func is_wieferich_prime(p, base=2) {
    powmod(base, p-1, p**2) == 1
}

say ("Wieferich primes less than 5000: ", 5000.primes.grep(is_wieferich_prime))
```

Output:

```
Wieferich primes less than 5000: [1093, 3511]
```

Wilson primes of order n

```
func is_wilson_prime(p, n = 1) {
    var m = p*p
    (factorialmod(n-1, m) * factorialmod(p-n, m) - (-1)**n) % m == 0
}

var primes = 1.1e4.primes

say "  n: Wilson primes\n——————————————————————"

for n in (1..11) {
    printf("%3d: %s\n", n, primes.grep {|p| is_wilson_prime(p, n) })
}
```

Output:

```
  n: Wilson primes
——————————————————————
  1: [5, 13, 563]
  2: [2, 3, 11, 107, 4931]
  3: [7]
  4: [10429]
  5: [5, 7, 47]
  6: [11]
  7: [17]
  8: []
  9: [541]
 10: [11, 1109]
 11: [17, 2713]
```

Window creation

Tk:

```
require('Tk')
%s'MainWindow'.new
%S'Tk'.MainLoop
```

Gtk2:

```
var gtk2 = require('Gtk2') -> init
var window = %s'Gtk2::Window'.new
window.signal_connect(destroy => { gtk2.main_quit })
window.show_all
gtk2.main
```

Wireworld

```
var f = [[], DATA.lines.map {['', .chars..., '']}..., []]

10.times {
    say f.map { .join(" ") + "\n" }.join
    var a = [[]]
    for y in (1 ..^ f.end) {
        var r = f[y]
        var rr = ['']
        for x in (1 ..^ r.end) {
            var c = r[x]
            rr << (
              given(c) {
                when('H') { 't' }
                when('t') { '.' }
                when('.') { <. H>[f.ft(y-1, y+1).map{.ft(x-1, x+1)...}.count('H') ~~ [1,2]] }
                default   { c }
              }
            )
        }
        rr << ''
        a << rr
    }
    f = [a..., []]
}

__DATA__
tH.........
.   .
   ...
.   .
Ht.. ......
```

Word break problem

```
func word_break (str, words) {

    var r = ->(str, arr=[]) {
        return true if str.is_empty
        for word in (words) {
            str.begins_with(word) || next
            if (__FUNC__(str.substr(word.len), arr)) {
                arr << word
                return arr
            }
        }
        return false
    }(str)

    r.kind_of(Array) ? r.reverse : nil
}

var words = %w(a o is pi ion par per sip miss able)
var strs = %w(a amiss parable opera operable inoperable permission mississippi)

for str in (strs) {
   printf("%11s: %s\n", str, word_break(str, words) \\ '(not possible)')
}
```

Output:

```
          a: ["a"]
      amiss: ["a", "miss"]
    parable: ["par", "able"]
      opera: ["o", "per", "a"]
   operable: ["o", "per", "able"]
 inoperable: (not possible)
 permission: ["per", "miss", "ion"]
mississippi: ["miss", "is", "sip", "pi"]
```

Word frequency

```
var count = Hash()
var file = File(ARGV[0] \\ '135-0.txt')

file.open_r.each { |line|
    line.lc.scan(/[\pL]+/).each { |word|
        count{word} := 0 ++
    }
}

var top = count.sort_by {|_,v| v }.last(10).flip

top.each { |pair|
    say "#{pair.key}\t-> #{pair.value}"
}
```

Output:

```
the     -> 41088
of      -> 19949
and     -> 14942
a       -> 14596
to      -> 13951
in      -> 11214
he      -> 9648
was     -> 8621
that    -> 7924
it      -> 6661
```

Word wrap

Greedy:

```
class String {
    method wrap(width) {
        var txt = self.gsub(/\s+/, " ")
        var len = txt.len
        var para = []
        var i = 0
        while (i < len) {
            var j = (i + width)
            while ((j < len) && (txt.char_at(j) != ' ')) { --j }
            para.append(txt.substr(i, j-i))
            i = j+1
        }
        return para.join("\n")
    }
}

var text = 'aaa bb cc ddddd'
say text.wrap(6)
```

Output:

```
aaa bb
cc
ddddd
```

Smart:

```
class SmartWordWrap {

    has width = 80

    method prepare_words(array, depth=0, callback) {

        var root = []
        var len = 0
        var i = -1

        var limit = array.end
        while (++i <= limit) {
            len += (var word_len = array[i].len)

            if (len > width) {
                if (word_len > width) {
```

```
                len -= word_len
                array.splice(i, 1, array[i].split(width)...)
                limit = array.end
                --i; next
            }
            break
        }

        root << [
            array.first(i+1).join(' '),
            self.prepare_words(array.ft(i+1), depth+1, callback)
        ]

        if (depth.is_zero) {
            callback(root[0])
            root = []
        }

        break if (++len >= width)
    }

    root
}

method combine(root, path, callback) {
    var key = path.shift
    path.each { |value|
        root << key
        if (value.is_empty) {
            callback(root)
        }
        else {
            value.each { |item|
                self.combine(root, item, callback)
            }
        }
        root.pop
    }
}

method wrap(text, width) {

    self.width = width
    var words = (text.kind_of(Array) ? text : text.words)

    var best = Hash(
        score => Inf,
        value => [],
    )

    self.prepare_words(words, callback: { |path|
        self.combine([], path, { |combination|
            var score = 0
            combination.ft(0, -2).each { |line|
                score += (width - line.len -> sqr)
            }

            if (score < best{:score}) {
                best{:score} = score
                best{:value} = []+combination
            }
        })
    })

    best{:value}.join("\n")
```

```
    }
}

var sww = SmartWordWrap();

var words = %w(aaa bb cc ddddd);
var wrapped = sww.wrap(words, 6);

say wrapped;
```

Output:

```
aaa
bb cc
ddddd
```

Write entire file

With error handling:

```
var file = File(__FILE__)
file.open_w(\var fh, \var err) || die "Can't open #{file}: #{err}"
fh.print("Hello world!")       || die "Can't write to #{file}: #{$!}"
```

Without error handling:

```
File(__FILE__).open_w.print("Hello world!")
```

Write float arrays to a text file

```
func writedat(filename, x, y, x_precision=3, y_precision=5) {
    var fh = File(filename).open_w

    for a,b in (x ~Z y) {
        fh.printf("%.*g\t%.*g\n", x_precision, a, y_precision, b)
    }

    fh.close
}

var x = [1, 2, 3, 1e11]
var y = x.map{.sqrt}

writedat('sqrt.dat', x, y)
```

Output:

```
1       1
2       1.4142
3       1.7321
1e+11   3.1623e+05
```

Write language name in 3D ASCII

```
var text = <<'EOT'

     ***
    *     *     *        **
    *           *       *
    *     *     *  ***  **
     ***  *  **** *   * *
        * * *   * ***** *
        * * *   * *     *
        * * *   * *     *
     ***  *  ****  ***  *
EOT

func banner3D(text, shift=-1) {
    var txt = text.lines.map{|line| line.gsub('*','__/').gsub(' ','   ')}
    var offset = txt.len.of {|i| " " * (shift.abs * i) }
    shift < 0 && offset.reverse!
    (offset »+« txt).join("\n")
}

say banner3D(text)
```

Output:

```
                __/__/__/
             __/              __/             __/                    __/__/
            __/                              __/                __/
           __/              __/             __/     __/__/__/     __/__/
             __/__/__/     __/     __/__/__/__/   __/        __/   __/
                    __/   __/   __/        __/   __/__/__/__/__/   __/
                   __/   __/   __/        __/   __/             __/
                  __/   __/   __/        __/   __/             __/
          __/__/__/     __/     __/__/__/__/     __/__/__/     __/
```

Xiaolin Wu's line algorithm

```
func plot(x, y, c) {
    c && printf("plot %d %d %.1f\n", x, y, c)
}

func fpart(x) {
    x - int(x)
}

func rfpart(x) {
    1 - fpart(x)
}

func drawLine(x0, y0, x1, y1) {

    var p = plot
    if (abs(y1 - y0) > abs(x1 - x0)) {
        p = {|arg| plot(arg[1, 0, 2]) }
        (x0, y0, x1, y1) = (y0, x0, y1, x1)
    }

    if (x0 > x1) {
        (x0, x1, y0, y1) = (x1, x0, y1, y0)
    }

    var dx = (x1 - x0)
    var dy = (y1 - y0)
    var gradient = (dy / dx)

    var xends = []
    var intery

    # handle the endpoints
    for x,y in [[x0, y0], [x1, y1]] {
        var xend = int(x + 0.5)
        var yend = (y + gradient*(xend-x))
        var xgap = rfpart(x + 0.5)

        var x_pixel = xend
        var y_pixel = yend.int
        xends << x_pixel

        p.call(x_pixel, y_pixel  , rfpart(yend) * xgap)
        p.call(x_pixel, y_pixel+1,  fpart(yend) * xgap)
        defined(intery) && next

        # first y-intersection for the main loop
        intery = (yend + gradient)
    }

    # main loop
    for x in (xends[0]+1 .. xends[1]-1) {
        p.call(x, intery.int,  rfpart(intery))
        p.call(x, intery.int+1, fpart(intery))
        intery += gradient
    }
}

drawLine(0, 1, 10, 2)
```

Output:

```
plot 0 1 0.5
plot 10 2 0.5
plot 1 1 0.9
plot 1 2 0.1
plot 2 1 0.8
plot 2 2 0.2
plot 3 1 0.7
plot 3 2 0.3
plot 4 1 0.6
plot 4 2 0.4
plot 5 1 0.5
plot 5 2 0.5
plot 6 1 0.4
plot 6 2 0.6
plot 7 1 0.3
plot 7 2 0.7
plot 8 1 0.2
plot 8 2 0.8
plot 9 1 0.1
plot 9 2 0.9
```

XML/DOM serialization

```
require('XML::Simple')
print %S'XML::Simple'.XMLout(
    :(root => :( element => 'Some text here' )),
    NoAttr => 1, RootName => '',
)
```

Output:

```
  <root>
    <element>Some text here</element>
  </root>
```

XML/Input

```
require('XML::Simple')

var ref = %S'XML::Simple'.XMLin('<Students>
  <Student Name="April" Gender="F" DateOfBirth="1989-01-02" />
  <Student Name="Bob" Gender="M"  DateOfBirth="1990-03-04" />
  <Student Name="Chad" Gender="M"  DateOfBirth="1991-05-06" />
  <Student Name="Dave" Gender="M"  DateOfBirth="1992-07-08">
    <Pet Type="dog" Name="Rover" />
  </Student>
  <Student DateOfBirth="1993-09-10" Gender="F" Name="&#x00C9;mily" />
</Students>')

ref{:Student}.each { say _{:Name} }
```

Output:

```
April
Bob
Chad
Dave
Émily
```

XML/Output

```
require('XML::Mini::Document')

var students = [
                ["April",         "Bubbly: I'm > Tam and <= Emily"],
                ["Tam O'Shanter", "Burns: \"When chapman billies leave the street ...\""],
                ["Emily",         "Short & shrift"]
               ]

var doc   = %O<XML::Mini::Document>.new
var root  = doc.getRoot
var studs = root.createChild("CharacterRemarks")

students.each { |s|
    var stud = studs.createChild("Character")
    stud.attribute("name", s[0])
    stud.text(s[1])
}

print doc.toString
```

XML/XPath

```
require('XML::XPath')

var x = %O<XML::XPath>.new(ARGF.slurp)

[x.findnodes('//item[1]')][0]
say [x.findnodes('//price')].map{x.getNodeText(_)}
[x.findnodes('//name')]
```

Output:

```
[14.5, 23.99, 4.95, 3.56]
```

XML validation

```
require('XML::LibXML')

func is_valid_xml(str, schema) {

    var parser    = %O<XML::LibXML>.new
    var xmlschema = %O<XML::LibXML::Schema>.new(string => schema)

    try {
        xmlschema.validate(parser.parse_string(str))
        true
    } catch {
        false
    }
}

var good_xml = '<a>5</a>'
var bad_xml  = '<a>5<b>foobar</b></a>'

var xmlschema_markup = <<'END'
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="a" type="xsd:integer"/>
</xsd:schema>
END

[good_xml, bad_xml].each { |xml|
    say "is_valid_xml(#{xml.dump}) : #{is_valid_xml(xml, xmlschema_markup)}"
}
```

Output:

```
is_valid_xml("<a>5</a>") : true
is_valid_xml("<a>5<b>foobar</b></a>") : false
```

Y combinator

```
var y = ->(f) {->(g) {g(g)}(->(g) { f(->(*args) {g(g)(args...)})})}

var fac = ->(f) { ->(n) { n < 2 ? 1 : (n * f(n-1)) } }
say 10.of { |i| y(fac)(i) }

var fib = ->(f) { ->(n) { n < 2 ? n : (f(n-2) + f(n-1)) } }
say 10.of { |i| y(fib)(i) }
```

Output:

```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Yin and yang

```
func circle (rad, cx, cy, fill='white', stroke='black') {
    say "<circle cx='#{cx}' cy='#{cy}' r='#{rad}' fill='#{fill}' stroke='#{stroke}' stroke-
width='1'/>"
}

func yin_yang (rad, cx, cy, fill='white', stroke='black', angle=90) {
    var (c, w) = (1, 0)
    angle != 0 && say "<g transform='rotate(#{angle}, #{cx}, #{cy})'>"
    circle(rad, cx, cy, fill, stroke)
    say("<path d='M #{cx} #{cy + rad}A #{rad/2} #{rad/2} 0 0 #{c} #{cx} #{cy} ",
        "#{rad/2} #{rad/2} 0 0 #{w} #{cx} #{cy - rad} #{rad} #{rad} 0 0 #{c} #{cx} ",
        "#{cy + rad} z' fill='#{stroke}' stroke='none' />")
    circle(rad/5, cx, cy + rad/2, fill, stroke)
    circle(rad/5, cx, cy - rad/2, stroke, fill)
    angle != 0 && say "</g>"
}

say '<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" xmlns:xlink="http://www.w3.org/1999/xlink">'

yin_yang(40, 50, 50)
yin_yang(20, 120, 120)

say '</svg>'
```

Zebra puzzle

```
var CONTENT = Hash(
            :House       => nil,
            :Nationality => [:English, :Swedish, :Danish, :Norwegian, :German],
            :Colour      => [:Red, :Green, :White, :Blue, :Yellow],
            :Pet         => [:Dog, :Birds, :Cats, :Horse, :Zebra],
            :Drink       => [:Tea, :Coffee, :Milk, :Beer, :Water],
            :Smoke       => [:PallMall, :Dunhill, :BlueMaster, :Prince, :Blend]
)

func adjacent(n,i,g,e) {
  (0..3).any {|x| (n[x]==i && g[x+1]==e) || (n[x+1]==i && g[x]==e) }
}

func leftof(n,i,g,e) {
  (0..3).any {|x| n[x]==i && g[x+1]==e }
}

func coincident(n,i,g,e) {
  n.indices.any {|x| n[x]==i && g[x]==e }
}

func solve {
  CONTENT{:Nationality}.permutations{|*nation|
    nation.first == :Norwegian ->
      && CONTENT{:Colour}.permutations {|*colour|
          leftof(colour,:Green,colour,:White) ->
       && coincident(nation,:English,colour,:Red) ->
       && adjacent(nation,:Norwegian,colour,:Blue) ->
       && CONTENT{:Pet}.permutations {|*pet|
             coincident(nation,:Swedish,pet,:Dog) ->
          && CONTENT{:Drink}.permutations {|*drink|
               drink[2] == :Milk ->
            && coincident(nation,:Danish,drink,:Tea) ->
            && coincident(colour,:Green,drink,:Coffee) ->
            && CONTENT{:Smoke}.permutations {|*smoke|
                coincident(smoke,:PallMall,pet,:Birds) ->
             && coincident(smoke,:Dunhill,colour,:Yellow) ->
             && coincident(smoke,:BlueMaster,drink,:Beer) ->
             && coincident(smoke,:Prince,nation,:German) ->
             && adjacent(smoke,:Blend,pet,:Cats) ->
             && adjacent(smoke,:Blend,drink,:Water) ->
             && adjacent(smoke,:Dunhill,pet,:Horse) ->
             && return [nation,colour,pet,drink,smoke]
} } } } } }

var res = solve();
var keys = [:House, :Nationality, :Colour, :Pet, :Drink, :Smoke]
var width = keys.map{ .len }
var fmt = width.map{|w| "%-#{w+2}s" }.join(" ")
say "The Zebra is owned by the man who is #{res[0][res[2].first_index(:Zebra)]}\n"
say (fmt % keys..., "\n", fmt % width.map{|w| "-"*w }...)
res[0].indices.map{|i| res.map{|a| a[i] }}.each_kv {|k,v| say fmt%(k,v...) }
```

Output:

```
The Zebra is owned by the man who is German

House   Nationality   Colour   Pet   Drink   Smoke
-----   -----------   ------   ---   -----   -----
0       Norwegian     Yellow   Cats  Water   Dunhill
1       Danish        Blue     Horse Tea     Blend
2       English       Red      Birds Milk    PallMall
3       German        Green    Zebra Coffee  Prince
4       Swedish       White    Dog   Beer    BlueMaster
```

Zeckendorf number representation

```
func fib(n) is cached {
    n < 2 ? 1
          : (fib(n-1) + fib(n-2))
}

func zeckendorf(n) {
    n == 0 && return '0'
    var i = 1
    ++i while (fib(i) <= n)
    gather {
        while (--i > 0) {
            var f = fib(i)
            f > n ? (take '0')
                  : (take '1'; n -= f)
        }
    }.join
}

for n (0..20) {
    printf("%4d: %8s\n", n, zeckendorf(n))
}
```

Output:

```
   0:        0
   1:        1
   2:       10
   3:      100
   4:      101
   5:     1000
   6:     1001
   7:     1010
   8:    10000
   9:    10001
  10:    10010
  11:    10100
  12:    10101
  13:   100000
  14:   100001
  15:   100010
  16:   100100
  17:   100101
  18:   101000
  19:   101001
  20:   101010
```

Zero to the zero power

```
[0, Complex(0, 0)].each {|n|
    say n**n
}
```

Output:

```
1
1
```

Taking the 0'th root of a number and raising it back to the zero power, we also get a 1:

```
say 0.root(0).pow(0)       # => 1
say ((0**(1/0))**0)        # => 1
```

Zhang-Suen thinning algorithm

```
class ZhangSuen(str, black="1") {
  const NEIGHBOURS = [[-1,0],[-1,1],[0,1],[1,1],[1,0],[1,-1],[0,-1],[-1,-1]]  # 8 neighbors
  const CIRCULARS = (NEIGHBOURS + [NEIGHBOURS.first])                          # P2, ... P9, P2

  has r = 0
  has image = [[]]

  method init {
    var s1 = str.lines.map{|line| line.chars.map{|c| c==black ? 1 : 0 }}
    var s2 = s1.len.of { s1[0].len.of(0) }
    var xr = range(1, s1.end-1)
    var yr = range(1, s1[0].end-1)
    do {
        r = 0
        xr.each{|x| yr.each{|y| s2[x][y] = (s1[x][y] - self.zs(s1,x,y,1)) }}  # Step 1
        xr.each{|x| yr.each{|y| s1[x][y] = (s2[x][y] - self.zs(s2,x,y,0)) }}  # Step 2
    } while !r.is_zero
    image = s1
  }

  method zs(ng,x,y,g) {
       (ng[x][y] == 0)                                      ->
    || (ng[x-1][y] + ng[x][y+1] + ng[x+g][y+g - 1] == 3) ->
    || (ng[x+g - 1][y+g] + ng[x+1][y] + ng[x][y-1] == 3) ->
    && return 0

    var bp1 = NEIGHBOURS.map {|p| ng[x+p[0]][y+p[1]] }.sum  # B(P1)
    return 0 if ((bp1 < 2) || (6 < bp1))

    var ap1 = 0
    CIRCULARS.map {|p| ng[x+p[0]][y+p[1]] }.each_cons(2, {|a,b|
        ++ap1 if (a < b)                                    # A(P1)
    })

    return 0 if (ap1 != 1)
    r = 1
  }

  method display {
    image.each{|row| say row.map{|col| col ? '#' : ' ' }.join }
  }
}

var text = <<EOS
00000000000000000000000000000000
01111111110000000111111110000000
01110001111000000111100111000000
01110000111000000111000011100000
01110001111000000111000000000000
01111111110000000111000000000000
01110111100000000111000011100000
01110011110011101111001111011100
01110001111011100111111110011100
00000000000000000000000000000000
EOS

ZhangSuen.new(text, black: "1").display
```

Output:

```
#######         ######
#     #        ##
#      #       #
#     #        #
##### #        #
     ##        #
      #    #   ##    ##   #
       #         ####
```

Zig-zag matrix

```
func zig_zag(w, h) {

    var r = []
    var n = 0

    h.of { |e|
        w.of { |f|
            [e, f]
        }
    }.reduce('+').sort { |a, b|
           (a[0]+a[1] <=> b[0]+b[1]) ||
           (a[0]+a[1] -> is_even ? a[0]<=>b[0]
                                : a[1]<=>b[1])
    }.each { |a|
       r[a[1]][a[0]] = n++
    }

    return r
}

zig_zag(5, 5).each { say .join('', {|i| "%4i" % i}) }
```

Output:

```
   0   1   5   6  14
   2   4   7  13  15
   3   8  12  16  21
   9  11  17  20  22
  10  18  19  23  24
```

Zumkeller numbers

```
func is_Zumkeller(n) {

    return false if n.is_prime
    return false if n.is_square

    var sigma = n.sigma

    # n must have an even abundance
    return false if (sigma.is_odd || (sigma < 2*n))

    # true if n is odd and has an even abundance
    return true if n.is_odd    # conjecture

    var divisors = n.divisors

    for k in (2 .. divisors.end) {
        divisors.combinations(k, {|*a|
            if (2*a.sum == sigma) {
                return true
            }
        })
    }

    return false
}

say "First 220 Zumkeller numbers:"
say (1..Inf -> lazy.grep(is_Zumkeller).first(220).join(' '))

say "\nFirst 40 odd Zumkeller numbers: "
say (1..Inf `by` 2 -> lazy.grep(is_Zumkeller).first(40).join(' '))

say "\nFirst 40 odd Zumkeller numbers not divisible by 5: "
say (1..Inf `by` 2 -> lazy.grep { _ % 5 != 0 }.grep(is_Zumkeller).first(40).join(' '))
```

Output:

```
First 220 Zumkeller numbers:
6 12 20 24 28 30 40 42 48 54 56 60 66 70 78 80 84 88 90 96 102 104 108 112 114 120 126 132 138 140 150

First 40 odd Zumkeller numbers:
945 1575 2205 2835 3465 4095 4725 5355 5775 5985 6435 6615 6825 7245 7425 7875 8085 8415 8505 8925 9135

First 40 odd Zumkeller numbers not divisible by 5:
81081 153153 171171 189189 207207 223839 243243 261261 279279 297297 351351 459459 513513 567567 621621
```

The end

To learn more about Sidef, see also:

- https://github.com/trizen/sidef
- https://github.com/trizen/sidef-scripts

Q&A

If you need help with Sidef, feel free to ask questions here:

- https://github.com/trizen/sidef/discussions/categories/q-a