COL

Table of Contents:

0: Intro

COL Stands for Call-based Object-oriented Language. This is because each statement in COL looks like a function call. For example, here's the code of a program that would output "Hello World!":

```
Sys:println("Hello World!")
```

Each statement is made up of three parts, separated by syntax characters. With this schema each statement boils down to

```
Host:function(arguments)
```

The host is either a library or an object such as a variable. The function is what to do with that object. The arguments are the parameters that tell that function what exactly to do.

If a function call has multiple arguments, they are separated by comas. If there aren't meant to be any parameters, the parentheses are left empty.

I: Variables

Variables are used for storing data in your program, so you can use it again later.

Creating a variable is done using the keyword is.
Var:is sets the variables value
Arguments: *value*
Example:

```
var_name:is(1)
```

Reading a variable's stored value can be done with the keyword val.
Var:val returns the variable's value
Arguments: None
Example:

```
var_name:val()
```

We can also pass code as an argument. For example, in the following code we output "Hello World!" again, but this time using a variable.

```
text:is("Hello World!")
Sys:println(text:val())
```

This leads to the following output:

```
-- Hello World!
```

You can obviously also assign one variable's value to another one:

```
x:is(144)
y:is(x:val())
Sys:println(y:val())
```

This leads to the following output:

```
-- 144
```

II: Basic types

Each value has a type. There are numbers, strings and booleans.

A number (`Num`) is any decimal number. For example:

```
x:is(3.2)
```

A string (`String`) is a string of characters. A string in code is surrounded by quotes. For example:

```
name:is("My Name")
```

A boolean (`Bool`) is either true or untrue. For example.

```
flag:is(untrue)
```

There are still a lot of other types that will be explained later. For now, here are some more examples:

```
Num    -> 13.684
          45
          25.7

String -> "abcdefgh"
          "ABC_dfg123942.42"
          "true"

Bool   -> true
          untrue
```

III: Math

There are numerous mathematical functions in the Math library (`M`) ranging from simple addition to calculating the square root of a number.

M:add Adds multiple values
Arguments: *values…*
Example:

```
Sys:println(M:add(2,3))
-- 5
```

M:sub Subtracts multiple values
Arguments: *values…*
Example:

```
Sys:println(M:sub(17,5,10))
-- 2
```

M:mul Multiplies multiple values
Arguments: *values…*
Example:

```
Sys:println(M:mul(7,8))
-- 56
```

M:div Divides multiple values
Arguments: *values…*
Example:

```
Sys:println(M:div(15,6))
-- 2.5
```

M:pow returns the base to the power of n
Arguments: *base, n*
Example:

```
Sys:println(M:pow(4,2))
-- 16
```

M:sqrt returns a number's square root
Arguments: *number*
Example:

```
Sys:println(M:sqrt(144))
-- 12
```

M:abs returns a number's absolute value
Arguments: *number*
Example:

```
Sys:println(M:abs(-37.9))
-- 37.9
```

M:mod returns the remainder of the number divided by the divisor
Arguments: *number, divisor*
Example:

```
Sys:println(M:mod(13,2))
-- 1
```

M:log2 returns the log 2 of a number
Arguments: *number*
Example:

```
Sys:println(M:log2(64))
-- 5
```

M:log10 returns the log 10 of a number
Arguments: *number*
Example:

```
Sys:println(M:log10(1000))
-- 3
```

M:log returns the log n of a number
Arguments: *number, n*
Example:

```
Sys:println(M:log(16, 4))
-- 2
```

M:round rounds a number
Arguments: *number*
Example:

```
Sys:println(M:round(48.2))
-- 48
```

M:floor rounds a number down
Arguments: *number*
Example:

```
Sys:println(M:floor(74,8))
-- 74
```

M:ceil rounds a number up
Arguments: *number*
Example:

```
Sys:println(M:ceil(31.2))
-- 32
```

M:sin sine of a number (in randians)
Arguments: *number*
Example:

```
x:is(M:sin(3.141))
```

M:cos cosine of a number (in randians)
Arguments: *number*
Example:

```
y:is(M:cos(26.7))
```

M:tan tangent of a number (in randians)
Arguments: *number*
Example:

```
z:is(M:tan(45))
```

M:asin arc sine of a number (in randians)
Arguments: *number*
Example:

```
x2:is(M:asin(1))
```

M:acos arc cosine of a number (in randians)
Arguments: *number*
Example:

```
y2:is(M:acos(0.5))
```

M:atan arc tangent of a number (in randians)
Arguments: *number*
Example:

```
z2:is(M:atan(-1))
```

M:pi returns pi
Arguments: None
Example:

```
Sys:println(M:pi())
-- 3.1415926…
```

M:e returns e
Arguments: None
Example:

```
e:is(M:e())
```

M:deg2rad converts a number from degrees to radians
Arguments: *number*
Example:

```
n_rad:is(M:deg2rad(90))
```

M:rad2deg converts a number from radians to degrees
Arguments: *number*
Example:

```
n_deg:println(M:rad2deg(3.14))
```

IV: Numbers

A stored number also has multiple functions.

Var:inc Increments the variable's value by the given value
Arguments: *value* (default 1)
Example:

```
x:is(5)
x:inc()
Sys:println(x:val())
-- 6
```

Var:dec Decrements the variable's value by the given value
Arguments: *value* (default 1)
Example:

```
x:is(25)
x:dec()
Sys:println(x:val())
-- 24
```

Var:mul Multiplies the variable's value by the given value
Arguments: *value* (default 1)
Example:

```
x:is(12)
x:mul(12)
Sys:println(x:val())
-- 144
```

Var:div Divides the variable's value by the given value
Arguments: *value* (default 1)
Example:

```
x:is(56)
x:div(8)
Sys:println(x:val())
-- 7
```

Var:string returns the variable's value as a string
Arguments: None
Example:

```
x:is(31)
Sys:println(x:string())
-- "31"
```

Num:maxval returns the highest possible value
Arguments: None
Example:

```
Sys:println(Num:maxval())
-- 1.7976931348623157e+308
```

Num:minval returns the lowest possible value
Arguments: None
Example:

```
Sys:println(Num:minval())
-- -1.7976931348623157e+308
```

Num:from Converts a string to a number
Arguments: *string*
Example:

```
Sys:println(Num:from("23"))
-- 23
```

V: Strings

String functions can be very useful in some situations:

Var:array returns an Array of the string's characters
Arguments: None
Example:

```
s:is("hello")
Sys:println(s:array())
-- <h, e, l, l, o>
```

Var:sp returns an Array of the string split by the given value
Arguments: *value* (default " ")
Example:

```
s:is("Hello how are you?")
Sys:printl(x:sp())
-- <Hello, how, are, you?>
```

Var:rp replaces all occurrences of target with value
Arguments: *target, value*
Example:

```
s:is("hello")
s:rp("e","a")
Sys:println(s:val())
-- hallo
```

Var:get returns the character at that index
Arguments: *index*
Example:

```
s:is("Apple")
Sys:printl(x:get(3))
-- l
```

Var:rev reverses the string
Arguments: None
Example:

```
s:is("hello")
Sys:println(s:rev())
-- olleh
```

Var:toNum returns the string as a number
Arguments: None
Example:

```
s:is("12.3")
Sys:printl(x:toNum())
-- 12.3
```

Var:len returns the length of the string
Arguments: None
Example:

```
s:is("Winter")
Sys:println(s:len())
-- 6
```

Var:has returns whether the string contains the given character
Arguments: *character*
Example:

```
Sys:printl("Peach":has("c"))
-- true
```

Var:sub returns the string from [start] to [end]
Arguments: *start* (default 0)*, end* (default length of the string)
Example:

```
s:is("Apple juice")
Sys:printl(x:sub(2,))
-- ple juice
```

Var:startswith returns whether the string starts with the given value
Arguments: *value*
Example:

```
s:is("hello")
Sys:println(s:startswith("h"))
-- true
```

Var:endswith returns whether the string ends with the given value
Arguments: *value*
Example:

```
s: :is("hello")
Sys:println(s:endswith("h"))
-- untrue
```

Var:strip removes given value from start and end of the string
Arguments: *value* (default " ")
Example:

```
Sys:printl(" Hello World ":strip())
-- Hello World
```

Var:rstrip removes given value from end of the string
Arguments: *value* (default " ")
Example:

```
s:is("Horizonnn")
Sys:println(s:rstrip("n"))
-- Horizo
```

Var:lstrip removes given value from start of the string
Arguments: *value* (default " ")
Example:

```
s:is("Apple")
Sys:printl(s:lstrip("A"))
-- pple
```

Var:fm formats the string with the given values
Arguments: *string, values…*
Example:

```
s:is(10)
Sys:printl("%1/%2":fm(8, s:val()))
-- 8/10
```

S:string converts the given value to a string
Arguments: *value*
Example:

```
Sys:println(S:string(12))
-- "12"
```

S:fm formats the string with the given values
Arguments: *string, values…*
Example:

```
s:is(3.14)
Sys:printl(S:fm("%1 is %2",
                "Pi", s:val()))
-- 12.3
```

S:ascii converts character to its Ascii code and the other way round
Arguments: *value*
Example:

```
Sys:printl(S:ascii(65))
-- A
```

S:sp returns an Array of the string split by the given value
Arguments: *string, value*
Example:

```
s:is("hello")
Sys:println(s:rp(s:val(),"e"))
-- <h, llo>
```

S:rp replaces all occurrences of target with value
Arguments: *string, target, value*
Example:

```
s:is("hello")
Sys:println(s:rp(s:val(),"e","a"))
-- hallo
```

S:rev reverses the string
Arguments: *string*
Example:

```
s:is("hello")
Sys:println(S:rev(s:val()))
-- olleh
```

VI: Booleans

Booleans contain all the logical operations.

Var:string converts the variable to a string
Arguments: None
Example:

```
b:is(untrue)
Sys:println(b:string())
-- "untrue"
```

B:and returns true if both values are true
Arguments: *bool, bool*
Example:

```
b:is(true)
Sys:printl(B:and(untrue, b:val()))
-- untrue
```

B:or returns true if either of the values is true
Arguments: *bool, bool*
Example:

```
b:is(true)
Sys:printl(B:or(untrue, b:val()))
-- true
```

B:xor returns true if one of the values is true and the other isn't
Arguments: *bool, bool*
Example:

```
b:is(true)
Sys:printl(B:xor(true, b:val()))
-- untrue
```

B:not the opposite of the current value
Arguments: *value*
Example:

```
b:is(untrue)
Sys:printl(B:not(b:val()))
-- true
```

B:all returns true if all given values are true
Arguments: *values…*
Example:

```
b1:is(true)
b2:is(true)
b3:is(untrue)
b3:is(true)
Sys:printl(B:all(b1:val(),b2:val(),
                 b3:val(),b4:val()))
-- untrue
```

B:any returns true if any of the given values are true
Arguments: *values…*
Example:

```
b1:is(untrue)
b2:is(true)
b3:is(untrue)
b3:is(untrue)
Sys:printl(B:any(b1:val(),b2:val(),
                 b3:val(),b4:val()))
-- true
```

VII: Arrays

An Array is basically a collection of values. You can access each of these values with its index. An Array's index starts with zero, so the first element of an array has the index 0.

Creating a new Array.

Ar:new generates an Array with the provided values
Arguments: *values…*
Example:

```
a:is(Ar:new(1,2,3,4))
Sys:println(a:val())
-- <1, 2, 3, 4>
```

Var:string returns the array as a string
Arguments: None
Example:

```
a:is(Ar:new(8,6,3,4.5))
Sys:println(a:val())
-- "<8, 6, 3, 4.5>"
```

Var:add adds an item to the end of the array
Arguments: *item*
Example:

```
a:is(Ar:new(1,2,3,4))
a:add(5)
Sys:println(a:val())
-- <1, 2, 3, 4, 5>
```

Var:insert inserts an item at a given index
Arguments: *index, item*
Example:

```
a:is(Ar:new(1,2,3,4))
a:insert(1,5)
Sys:println(a:val())
-- <1, 5, 2, 3, 4>
```

Var:add_ref adds a reference of the item to the end of the Array
Arguments: *item*
Example:

```
a:is(Ar:new(1,2,3,4))
x:is(5)
a:add_ref(x:val())
Sys:println(a:val())
x:is(12)
Sys:println(a:val())
-- <1, 2, 3, 4, 5>
-- <1, 2, 3, 4, 12>
```

Var:rem removes the first occurrence of the given item
Arguments: *item*
Example:

```
a:is(Ar:new(1,2,4,6,2,9))
a:rem(2)
Sys:println(a:val())
-- <1, 4, 6, 2, 9>
```

Var:get returns the item at that index
Arguments: *index*
Example:

```
 a:is(Ar:new(1,2,4,6,12,30))
Sys:println(a:get(2))
 -- 4
```

Var:set sets the item at the given index
Arguments: *index, item*
Example:

```
 a:is(Ar:new(1,2,3,4))
 a:set(1,5)
 Sys:println(a:val())
 -- <1, 5, 3, 4>
```

Var:len returns the length of the array
Arguments: None
Example:

```
 a:is(Ar:new(1,2,3,4,5,6))
 Sys:println(a:len())
 -- 6
```

Var:has returns whether the given item is in the array
Arguments: *item*
Example:

```
 a:is(Ar:new(1,2,3,4,5,6))
 Sys:println(a:has(3))
 -- true
```

Var:join joins all items of the array with the given separator
Arguments: *separator*
Example:

```
a:is(Ar:new(1,2,3,4,5,6))
Sys:println(a:join("-"))
-- "1-2-3-4-5-6"
```

Var:map Executes the given code for each item in the array
Arguments: *name, code*
Example:

```
a:is(Ar:new(1,2,3,4,5))
Sys:println(a:map(x,
  M:mul(2,x:val())))
-- <2, 4, 6, 8, 10>
```

Var:filter removes all items that don't return true when the given code is executed with their value
Arguments: *name, code*
Example:

```
a:is(Ar:new(1,2,3,4,5,6,7))
Sys:println(a:filter(x,Op:eq(0,
  M:mod(x:val(),2))))
-- <2, 4, 6>
```

Var:all returns whether all items return true when the given code is executed with their value
Arguments: *name, code*
Example:

```
a:is(Ar:new(1,2,3,4,5,6,7))
Sys:println(a:all(x,Op:eq(0,
  M:mod(x:val(),2))))
-- untrue
```

Var:any returns whether any items return true when the given code is executed with their value

Arguments: *name, code*

Example:

```
a:is(Ar:new(1,2,3,4,5,6,7))
Sys:println(a:any(x,Op:eq(0,
  M:mod(x:val(),2))))
-- true
```

Var:min returns the item that returns the lowest value when executing the given code with its value

Arguments: *name, code*

Example:

```
a:is(Ar:new(1,2,3,4,5))
Sys:println(a:min(x,
  M:mul(-1,x:val())))
-- 5
```

Var:max returns the item that returns the highest value when executing the given code with its value

Arguments: *name, code*

Example:

```
a:is(Ar:new(1,2,3,4,5))
Sys:println(a:max())
-- 5
```

Var:sum returns the sum of the values the code returns for all items

Arguments: *name, code*

Example:

```
a:is(Ar:new(1,2,3,4,5))
Sys:println(a:sum(x,
  M:mul(x:val(),2)))
-- 30
```

Var:rev returns the reversed array
executing the given code with its value
Arguments: None
Example:

```
a:is(Ar:new(1,2,3,4,5))
Sys:println(a:rev())
-- <5, 4, 3, 2, 1>
```

Var:sort sorts it by the given criteria
Arguments: *name…, code*
Example:

```
a:is(Ar:new(5,1,4,2,3))
Sys:println(a:sort())
-- <1, 2, 3, 4, 5>
```

VIII: Operators

Operators are used to compare two things.

Op:eq returns if two values are equal
Arguments: *value, value*
Example:

```
x:is(3)
Sys:println(Op:eq(x:val(),3))
-- true
```

Op:uneq returns if two values are not equal
Arguments: *value, value*
Example:

```
x:is(3)
Sys:println(Op:uneq(x:val(),3))
-- untrue
```

Op:gt returns if value is greater than another one
Arguments: *value, value*
Example:

```
x:is(3)
Sys:println(Op:gt(x:val(),5))
-- untrue
```

Op:lt returns if value is less than another one
Arguments: *value, value*
Example:

```
x:is(3)
Sys:println(Op:lt(x:val(),5))
-- true
```

Op:ge returns if value is greater or equal to another one
Arguments: *value, value*
Example:

```
x:is(7)
Sys:println(Op:gt(x:val(),7))
-- true
```

Op:le returns if value is less or equal to another one
Arguments: *value, value*
Example:

```
x:is(3)
Sys:println(Op:lt(x:val(),2))
-- untrue
```

IX: Expressions

Expressions are used to execute different lines of code depending on some criteria.

Exp:if evaluates a condition and does something accordingly
Arguments: *condition, then, [else]*
Example:

```
x:is(12)
Exp:if(Op:eq(12,x:val()),
  Sys:println("Yes"),
  Sys:println("No")
)
-- Yes
```

Exp:while repeats the code as long as the condition is true
Arguments: *condition, code*
Example:

```
a:is(3)
Exp:while(Op:gt(a:val(),0,
  Sys:println(a:val())
  a:dec()
)
-- 3
-- 2
-- 1
```

Exp:for (with 4 arguments) defines a variable. While the condition is True, executes code and then runs increase.

Arguments: *declaration, condition, increase, code*

Example:

```
Exp:for(i:is(1),Op:lt(i:val(),6),
                              i:inc(),
  Sys:println(i:val())
)
-- 1
-- 2
-- 3
-- 4
-- 5
```

Exp:for (with 3 arguments) loops through a collection and assigns each item to a variable named name, then runs the code

Arguments: *name, collection, code*

Example:

```
a:is(Ar:new(1,2,3,4,5,6,7))
Exp:for(x,a:val(),
  Sys:println(x:val())
)
-- 1
-- 2
-- 3
-- 4
-- 5
-- 6
-- 7
```

Exp:break breaks out of the current loop
Arguments: None
Example:

```
a:is(3)
Exp:while(Op:gt(a:val(),0,
  Exp:if(Op:eq(2,a:val()),
    Exp:break()
  )
  Sys:println(a:val())
  a:dec()
)
-- 3
```

Exp:continue stops the loop and continues with the next iteration
Arguments: None
Example:

```
a:is(3)
Exp:while(Op:gt(a:val(),0,
  Exp:if(Op:eq(2,a:val()),
    Exp:continue()
  )
  Sys:println(a:val())
  a:dec()
)
-- 3
-- 1
```

Var:return returns the given value
Arguments: *[value]*
Example:

```
f:func(x,
  Exp:return(M:add(x:val(),12.2))
)
a:is(f:run(10))
Sys:println(a:val())
-- 22.2
```

X: Functions

A function is a bit of code that you can call with different parameters. It's like a variable but instead of storing values, it stores code.

FName:func creates a new function
Arguments: *[parameters…], code*
Example:

```
f:func(x,
  Sys:print(x:val())
  Sys:println("!")
)
f:run("Hello World")
-- Hello World!
```

FName:run runs the function with the given arguments
Arguments: *[arguments…]*
Example:

```
f:func(x,y,
  Sys:print(x:val())
  Sys:println(y:val())
)
f:run("Hello World","!")
f:run("Function","?")
-- Hello World!
-- Function?
```

XI: Classes

A class is like a custom type with custom functions for that type.

CName:Class creates a new class
Arguments: *definition*
Example:

```
person:class(
)
```

In the definition of a class there are multiple keywords.

name:var creates a member variable with value as the default value
Arguments: *[value]*
Example:

```
person:class(
   name:var()
   age:var(1)
)
```

name:pvar creates a private variable with value as the default value.
Private variables can only be accessed from inside the class
Arguments: *[value]*
Example:

```
person:class(
   name:var()
   age:var(1)
   hobby:pvar()
)
```

name:func creates a member function with the given values
Arguments: *[parameters…], code*
Example:

```
person:class(
   name:var()
   age:var(1)
   hobby:pvar()
   birthday:func(,age:inc())
)
```

name:pfunc creates a private function with the given values
Arguments: *[parameters…], code*
Example:

```
person:class(
   name:var()
   age:var(1)
   hobby:pvar()
   birthday:func(,age:inc())
   do_hobby:pfunc(time,...)
)
```

There are also multiple built-in functions you can define.

…new:func the method that is called to an instance of this class
Arguments: *[parameters…], code*
Example:

```
person:class(
   name:var()
   age:var(1)
   hobby:pvar()
   birthday:func(,age:inc())
   do_hobby:pfunc(time,...)
   …new:func(n,a,
     name:is(n:val())
     age:is(a:val())
   )
)
```

…string:func the function that is called when converting to a string
Arguments: *[value]*
Example:

```
person:class(
   name:var()
   age:var(1)
   ...
   …string:func(,
     Exp:return("%1 (%2)":fm(
      name:val(),age:val()
     ))
   )
)
```

…add:func adding to this class
Arguments: *[parameters…], code*

…sub:func adding to this class
Arguments: *[parameters…], code*

…mul:func multiplying this class
Arguments: *[parameters…], code*

…div:func dividing this class
Arguments: *[parameters…], code*

…mod:func modulo of this class
Arguments: *[parameters…], code*

…array:func converting this class to an array (used for Exp:for)
Arguments: *[parameters…], code*

…toNum:func converting this class to a Number (used for Num:from)
Arguments: *[parameters…], code*

…eq:func comparing
Arguments: *[parameters…], code*

…hash:func hashing this class
Arguments: *[parameters…], code*

XII: Sys

Sys:println prints a value to the console with a newline at the end
Arguments: *value*
Example:

```
Sys:println("Hello World!")
Sys:println("Hiii <3")
-- Hello World!
-- Hiii <3
```

Sys:println prints a value to the console
Arguments: *value*
Example:

```
Sys:print("Hello World!")
Sys:println("Hiii <3")
-- Hello World!Hiii <3
```

Sys:input reads input from the console, displays the given value
Arguments: *[value]*
Example:

```
s:is()Sys:input("Name: "))
Sys:println(s:val())
-- Name: Clara
-- Clara
```

Sys:import executes the code the in the give file
Arguments: *file*
Example:

```
Sys:import("hello_world.col")
...
```

Sys:hash hashes the given values
Arguments: *values…*
Example:

```
h:is(Sys:hash("Hello World!"))
```

Sys:include imports a defined library
Arguments: *value*
Example:

```
Sys:include(IO)
```

XIII: File-IO

The IO module can be imported using Sys:include(IO)

IO:reader returns an instance of the class Reader of filename.
Arguments: *filename*
Example:

```
r:is(IO:reader("file.txt"))
```

Reader:readline reads the next line in the file
Arguments: None
Example:

```
r:is(IO:reader("file.txt"))
Sys:println(r:readline())
-- ...
```

Reader:readline reads all lines and returns them as an Array
Arguments: None
Example:

```
r:is(IO:reader("file.txt"))
Sys:println(r:readall())
-- <...>
```

Reader:done returns whether there are more lines to read.
Arguments: None
Example:

```
r:is(IO:reader("file.txt"))
Exp:while(B:not(r:done()),
  Sys:println(r:readline())
)
-- ...
```

Reader:close closes the file.
Arguments: None
Example:

```
r:is(IO:reader("file.txt"))
Exp:while(B:not(r:done()),
  Sys:println(r:readline())
)
r:close()
-- ...
```

IO:read creates a Reader named name for the file filename and closes it after executing code.
Arguments: *name, filename, code*
Example:

```
IO:read(r,"file.txt",
  Exp:while(B:not(r:done()),
    Sys:println(r:readline())
  )
)
-- ...
```

IO:writer returns an instance of the class Writer of filename. Append determines if the current content of the file is deleted
Arguments: *filename, append* (default: untrue)
Example:

```
r:is(IO:Writer("file.txt"),true)
-- ...
```

Writer:writeline writes the text to the file with a newline at the end
Arguments: *text*
Example:

```
r:is(IO:reader("file.txt"))
r:writeline("Hello World!")
```

Writer:write writes the text to the file
Arguments: *text*
Example:

```
r:is(IO:reader("file.txt"))
r:write("Hello World!")
```

Writer:close closes the file
Arguments: None
Example:

```
r:is(IO:reader("file.txt"))
r:writeline("Hello World!")
r:close()
```

IO:read creates a Writer named name for the file filename and closes it after executing code.
Arguments: *name, filename, append, code*
Example:

```
IO:write(r,"file.txt",
  r:write("Hell")
  r:writeline("o World!")
)
```

XIV: Dictionarys

Dict:new creates a new Dictionary
Arguments: None
Example:

```
d:is(Dict:new())
```

Var:set sets the value for the given key
Arguments: *key, value*
Example:

```
d:is(Dict:new())
d:set(1,0)
d:set(2,17)
Sys:println(d:val())
-- <{(1 -> 0), (2 -> 17)}>
```

Var:get reads all lines and returns them as an Array
Arguments: *key*
Example:

```
d:is(Dict:new())
d:set(1,0)
d:set(2,17)
Sys:println(d:get(1))
-- 0
```

Var:has returns whether there are more lines to read.
Arguments: *key*
Example:

```
d:is(Dict:new())
d:set(1,0)
Sys:println(d:has(1))
-- true
```

Var:string returns the dictionary to a string
Arguments: *key*
Example:

```
d:is(Dict:new())
d:set(1,0)
d:set(2,17)
Sys:println(d:string())
-- "<{(1 -> 0), (2 -> 17)}>"
```

Var:array returns the dictionary as an array of valuepairs
Arguments: *key*
Example:

```
d:is(Dict:new())
d:set(1,0)
Sys:println(d:array())
-- <1 -> 0, 2 -> 17>
```

ValuePair:new creates a new ValuePair
Arguments: *key, value*
Example:

```
v:is(ValuePair:new(1,0))
Sys:println(v:val())
-- 1 -> 0
```

Var:key returns the key of the valuepair
Arguments: None

Var:value returns the value of the valuepair
Arguments: None

Var:string returns the valuepair as a string
Arguments: None

XV: Librarys

A custom library can be created using the lib keyword

LName:lib creates new library named name
Arguments: *Declaration*
Example:

```
d:is(Dict:new())
```

In the definition:

name:class defines a class
Arguments: *Definition*

FName:func defines a function that can be called with LName:FName
Arguments: *[arguments...], code*
Example:

```
library:lib(
  f:func(,Sys:println("<3"))
)
library:f()
-- <3
```