Self is a fast prototype based OO language which runs in its own JIT vm. Most development is done through interacting with live objects through a visual development environment called *morphic* with integrated browsers and debugger.

Everything in Self is an object. All computation is done by sending messages to objects. Objects in Self can be understood as sets of key-value slots.

Constructing objects

The inbuild Self parser can construct objects, including method objects.

```
"This is a comment"

"A string:"
'This is a string with \'escaped\' characters.\n'

"A 30 bit integer"
23

"A 30 bit float"
3.2

"-20"
-14r16

"An object which only understands one message, 'x' which returns 20"
(|
  x = 20.
|)

"An object which also understands 'x:' which sets the x slot"
(|
  x <- 20.
|)

"An object which understands the method 'doubleX' which
doubles the value of x and then returns the object"
(|
  x <- 20.
  doubleX = (x: x * 2. self)
|)

"An object which understands all the messages
that 'traits point' understands". The parser
looks up 'traits point' by sending the messages
'traits' then 'point' to a known object called
the 'lobby'. It looks up the 'true' object by
also sending the message 'true' to the lobby."
(|     parent* = traits point.
       x = 7.
       y <- 5.
       isNice = true.
|)
```

Sending messages to objects

Messages can either be unary, binary or keyword. Precedence is in that order. Unlike Smalltalk, the precedence of binary messages must be specified, and all keywords after the first must start with a capital letter. Messages are separeated from their destination by whitespace.

```
"unary message, sends 'printLine' to the object '23'
which prints the string '23' to stdout and returns the receiving object (ie 23)"
23 printLine

"sends the message '+' with '7' to '23', then the message '*' with '8' to the result"
(23 + 7) * 8

"sends 'power:' to '2' with '8' returns 256"
2 power: 8

"sends 'keyOf:IfAbsent:' to 'hello' with arguments 'e' and '-1'.
Returns 1, the index of 'e' in 'hello'."
'hello' keyOf: 'e' IfAbsent: -1
```

Blocks

Self defines flow control like Smalltalk and Ruby by way of blocks. Blocks are delayed computations of the form:

```
[|:x. localVar| x doSomething with: localVar]
```

Examples of the use of a block:

```
"returns 'HELLO'"
'hello' copyMutable mapBy: [|:c| c capitalize]

"returns 'Nah'"
'hello' size > 5 ifTrue: ['Yay'] False: ['Nah']

"returns 'HaLLO'"
'hello' copyMutable mapBy: [|:c|
   c = 'e' ifTrue: [c capitalize]
            False: ['a']]
```

Multiple expressions are separated by a period. ^ returns immediately.

```
"returns An 'E'! How icky!"
'hello' copyMutable mapBy: [|:c. tmp <- ''|
   tmp: c capitalize.
   tmp = 'E' ifTrue: [^ 'An \'E\'! How icky!'].
   c capitalize
   ]
```

Blocks are performed by sending them the message 'value' and inherit (delegate to) their contexts:

```
"returns 0"
[|x|
    x: 15.
    "Repeatedly sends 'value' to the first block while the result of sending 'value' to the
     second block is the 'true' object"
    [x > 0] whileTrue: [x: x - 1].
    x
] value
```

Methods

Methods are like blocks but they are not within a context but instead are stored as values of slots. Unlike Smalltalk, methods by default return their final value not 'self'.

```
"Here is an object with one assignable slot 'x' and a method 'reduceXTo: y'.
Sending the message 'reduceXTo: 10' to this object will put
the object '10' in the 'x' slot and return the original object"
(|
    x <- 50.
    reduceXTo: y = (
        [x > y] whileTrue: [x: x - 1].
        self)
|)
.
```

Prototypes

Self has no classes. The way to get an object is to find a prototype and copy it.

```
| d |
d: dictionary copy.
d at: 'hello' Put: 23 + 8.
d at: 'goodbye' Put: 'No!.
"Prints No!"
( d at: 'goodbye' IfAbsent: 'Yes! ) printLine.
"Prints 31"
( d at: 'hello' IfAbsent: -1 ) printLine.
```

Further information

The Self handbook has much more information, and nothing beats hand-on experience with Self by downloading it from the homepage.