Pogoscript is a little language that emphasises readability, DSLs and provides excellent asynchronous primitives for writing connected JavaScript applications for the browser or server.

```
// defining a variable
water temperature = 24

// re-assigning a variable after its definition
water temperature := 26

// functions allow their parameters to be placed anywhere
temperature at (a) altitude = 32 - a / 100

// longer functions are just indented
temperature at (a) altitude :=
    if (a < 0)
        water temperature
    else
        32 - a / 100

// calling a function
current temperature = temperature at 3200 altitude

// this function constructs a new object with methods
position (x, y) = {
    x = x
    y = y

    distance from position (p) =
        dx = self.x - p.x
        dy = self.y - p.y
        Math.sqrt (dx * dx + dy * dy)
}

// `self` is similar to `this` in JavaScript with the
// exception that `self` isn't redefined in each new
// function definition
// `self` just does what you expect

// calling methods
position (7, 2).distance from position (position (5, 1))

// as in JavaScript, objects are hashes too
position.'x' == position.x == position.('x')

// arrays
positions = [
    position (1, 1)
    position (1, 2)
    position (1, 3)
]

// indexing an array
positions.0.y
```

```
n = 2
positions.(n).y

// strings
poem = 'Tail turned to red sunset on a juniper crown a lone magpie cawks.
        Mad at Oryoki in the shrine-room -- Thistles blossomed late afternoon.
        Put on my shirt and took it off in the sun walking the path to lunch.
        A dandelion seed floats above the marsh grass with the mosquitos.
        At 4 A.M. the two middleaged men sleeping together holding hands.
        In the half-light of dawn a few birds warble under the Pleiades.
        Sky reddens behind fir trees, larks twitter, sparrows cheep cheep cheep
        cheep cheep.'

// that's Allen Ginsburg

// interpolation
outlook = 'amazing!'
console.log "the weather tomorrow is going to be #(outlook)"

// regular expressions
r/(\d+)m/i
r/(\d+) degrees/mg

// operators
true @and true
false @or true
@not false
2 < 4
2 >= 2
2 > 1

// plus all the javascript ones

// to define your own
(p1) plus (p2) =
    position (p1.x + p2.x, p1.y + p2.y)

// `plus` can be called as an operator
position (1, 1) @plus position (0, 2)
// or as a function
(position (1, 1)) plus (position (0, 2))

// explicit return
(x) times (y) = return (x * y)

// new
now = @new Date ()

// functions can take named optional arguments
spark (position, color: 'black', velocity: {x = 0, y = 0}) = {
    color = color
    position = position
    velocity = velocity
}
```

```
red = spark (position 1 1, color: 'red')
fast black = spark (position 1 1, velocity: {x = 10, y = 0})

// functions can unsplat arguments too
log (messages, ...) =
    console.log (messages, ...)

// blocks are functions passed to other functions.
// This block takes two parameters, `spark` and `c`,
// the body of the block is the indented code after the
// function call

render each @(spark) into canvas context @(c)
    ctx.begin path ()
    ctx.stroke style = spark.color
    ctx.arc (
        spark.position.x + canvas.width / 2
        spark.position.y
        3
        0
        Math.PI * 2
    )
    ctx.stroke ()

// asynchronous calls

// JavaScript both in the browser and on the server (with Node.js)
// makes heavy use of asynchronous IO with callbacks. Async IO is
// amazing for performance and making concurrency simple but it
// quickly gets complicated.
// Pogoscript has a few things to make async IO much much easier

// Node.js includes the `fs` module for accessing the file system.
// Let's list the contents of a directory

fs = require 'fs'
directory listing = fs.readdir! '.'

// `fs.readdir()` is an asynchronous function, so we can call it
// using the `!` operator. The `!` operator allows you to call
// async functions with the same syntax and largely the same
// semantics as normal synchronous functions. Pogoscript rewrites
// it so that all subsequent code is placed in the callback function
// to `fs.readdir()`.

// to catch asynchronous errors while calling asynchronous functions

try
    another directory listing = fs.readdir! 'a-missing-dir'
catch (ex)
    console.log (ex)

// in fact, if you don't use `try catch`, it will raise the error up the
```

```
// stack to the outer-most `try catch` or to the event loop, as you'd expect
// with non-async exceptions

// all the other control structures work with asynchronous calls too
// here's `if else`
config =
    if (fs.stat! 'config.json'.is file ())
        JSON.parse (fs.read file! 'config.json' 'utf-8')
    else
        {
            color: 'red'
        }

// to run two asynchronous calls concurrently, use the `?` operator.
// The `?` operator returns a *future* which can be executed to
// wait for and obtain the result, again using the `!` operator

// we don't wait for either of these calls to finish
a = fs.stat? 'a.txt'
b = fs.stat? 'b.txt'

// now we wait for the calls to finish and print the results
console.log "size of a.txt is #(a!.size)"
console.log "size of b.txt is #(b!.size)"

// futures in Pogoscript are analogous to Promises
```

That's it.

Download Node.js and `npm install pogo`.

There is plenty of documentation on http://pogoscript.org/, including a cheat sheet, a guide, and how Pogoscript translates to Javascript. Get in touch on the google group if you have questions!