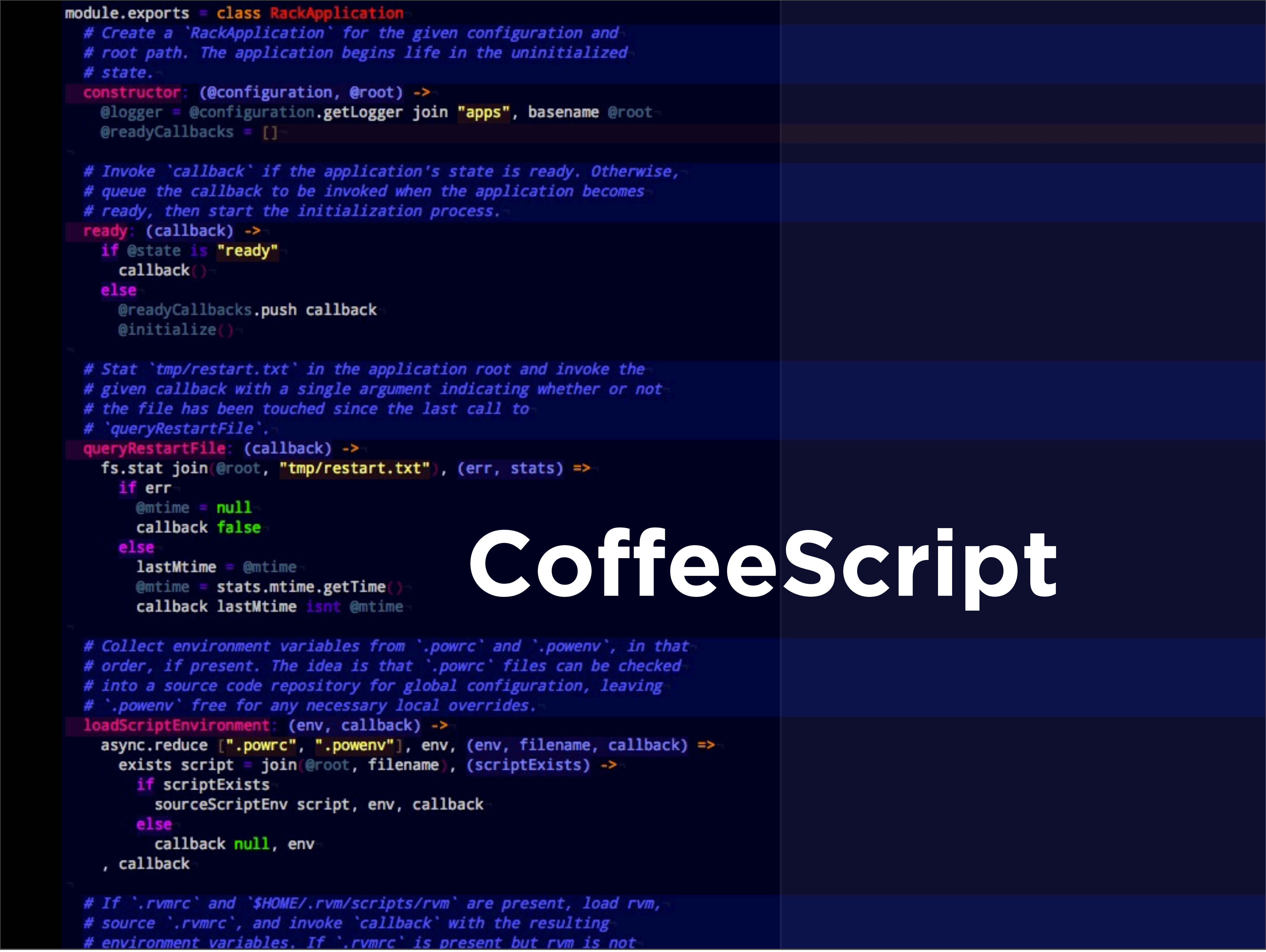
CoffeeScript



```coffeescript
module.exports = class RackApplication
  # Create a `RackApplication` for the given configuration and
  # root path. The application begins life in the uninitialized
  # state.
  constructor: (@configuration, @root) ->
    @logger = @configuration.getLogger join "apps", basename @root
    @readyCallbacks = []

  # Invoke `callback` if the application's state is ready. Otherwise,
  # queue the callback to be invoked when the application becomes
  # ready, then start the initialization process.
  ready: (callback) ->
    if @state is "ready"
      callback()
    else
      @readyCallbacks.push callback
      @initialize()

  # Stat `tmp/restart.txt` in the application root and invoke the
  # given callback with a single argument indicating whether or not
  # the file has been touched since the last call to
  # `queryRestartFile`.
  queryRestartFile: (callback) ->
    fs.stat join(@root, "tmp/restart.txt"), (err, stats) =>
      if err
        @mtime = null
        callback false
      else
        lastMtime = @mtime
        @mtime = stats.mtime.getTime()
        callback lastMtime isnt @mtime

  # Collect environment variables from `.powrc` and `.powenv`, in that
  # order, if present. The idea is that `.powrc` files can be checked
  # into a source code repository for global configuration, leaving
  # `.powenv` free for any necessary local overrides.
  loadScriptEnvironment: (env, callback) ->
    async.reduce [".powrc", ".powenv"], env, (env, filename, callback) =>
      exists script = join(@root, filename), (scriptExists) ->
        if scriptExists
          sourceScriptEnv script, env, callback
        else
          callback null, env
    , callback

  # If `.rvmrc` and `$HOME/.rvm/scripts/rvm` are present, load rvm,
  # source `.rvmrc`, and invoke `callback` with the resulting
  # environment variables. If `.rvmrc` is present but rvm is not
```

The State of CoffeeScript

“It’s just JavaScript”

Under the Hood

```
Scope.prototype.find = function(name, options) {
  if (this.check(name, options)) {
    return true;
  }
  this.add(name, 'var');
  return false;
};
```

Syntax + Semantics + Goodies

```
var square = function(x) {
  return x * x;
};
```

```
# Functions...

square = (x) -> x * x
```

```
# Literals.


stooges = ['Moe', 'Curly', 'Larry']


elements = {
  hydrogen: 1,
  silicon:  14,
  uranium:  92
}
```

```
# Whitespace for blocks.


today = "Monday"


if today is "Monday"
  console.log "Strange Loop!"
else
  console.log "Awww..."
```

```
# Array, Range, and Object comprehensions.


list = ['a', 'b', 'c']

console.log "Hi " + letter for letter in list

console.log("Hi " + letter for letter in list)


for name of process
  console.log name


for i in [0..10]
  console.log i
```

```
# Even conditional statements.

console.log if false
  100
else
  200

# Even crazy things like a try/catch.

tryResult = try
  missing.object
catch err
  "And the error is: #{err}"

console.log tryResult
```

```
bestActor = (winner, others...) ->
  console.log "And the Oscar goes to ... #{winner}!"
  console.log "(with #{others.length} runners up)"

bestActor 'Gypo Nolan', 'Clark Gable',
  'Paul Muni', 'Ludwig Satz'
```

```
# The existential operator.

sue =
  name: 'Sue Jones'
  age:  27

console.log sue.name.length

console.log sue.non?.existent.property

console.log sue.name.split('').reverse?()
```

```
# Class literals.

shanty = """
         Now let every man drink off his full bumper,
         And let every man drink off his full glass;
         We will drink and be jolly and drown melancholy,
         And heres to the health of each true-hearted lass.
         """

class Mariner
  sing: -> console.log shanty

class UptightSailor extends Mariner
  sing: -> console.log "I'd rather not."

class Pirate extends Mariner
  sing: ->
    super()
    console.log 'AAAAARRRRRRRRRRRR!'

new Pirate().sing()
```

```
class Person

  constructor: (@name) ->
    # body of constructor.

  introduce: =>
    console.log "Hi, I'm #{@name}."


groucho = new Person "Groucho Marx"

sayHi = groucho.introduce

sayHi()
```

```
futurists =
  sculptor: "Umberto Boccioni"
  painter:  "Vladimir Burliuk"
  poet:
    name:   "F.T. Marinetti"
    address: [
      "Via Roma 42R"
      "Bellagio, Italy 22021"
    ]

{poet: {name: name, address: [street, city]}} = futurists

console.log "#{name} lives on #{street}."
```

```
# David's web server example:

http = require 'http'

server = http.createServer (req, res) ->
  res.writeHead 200, 'Content-Type': 'text/plain'
  res.end 'Ahoy, Strange Loop!'

server.listen 3000
console.log 'Listening on 3000'
```

# Design Decisions

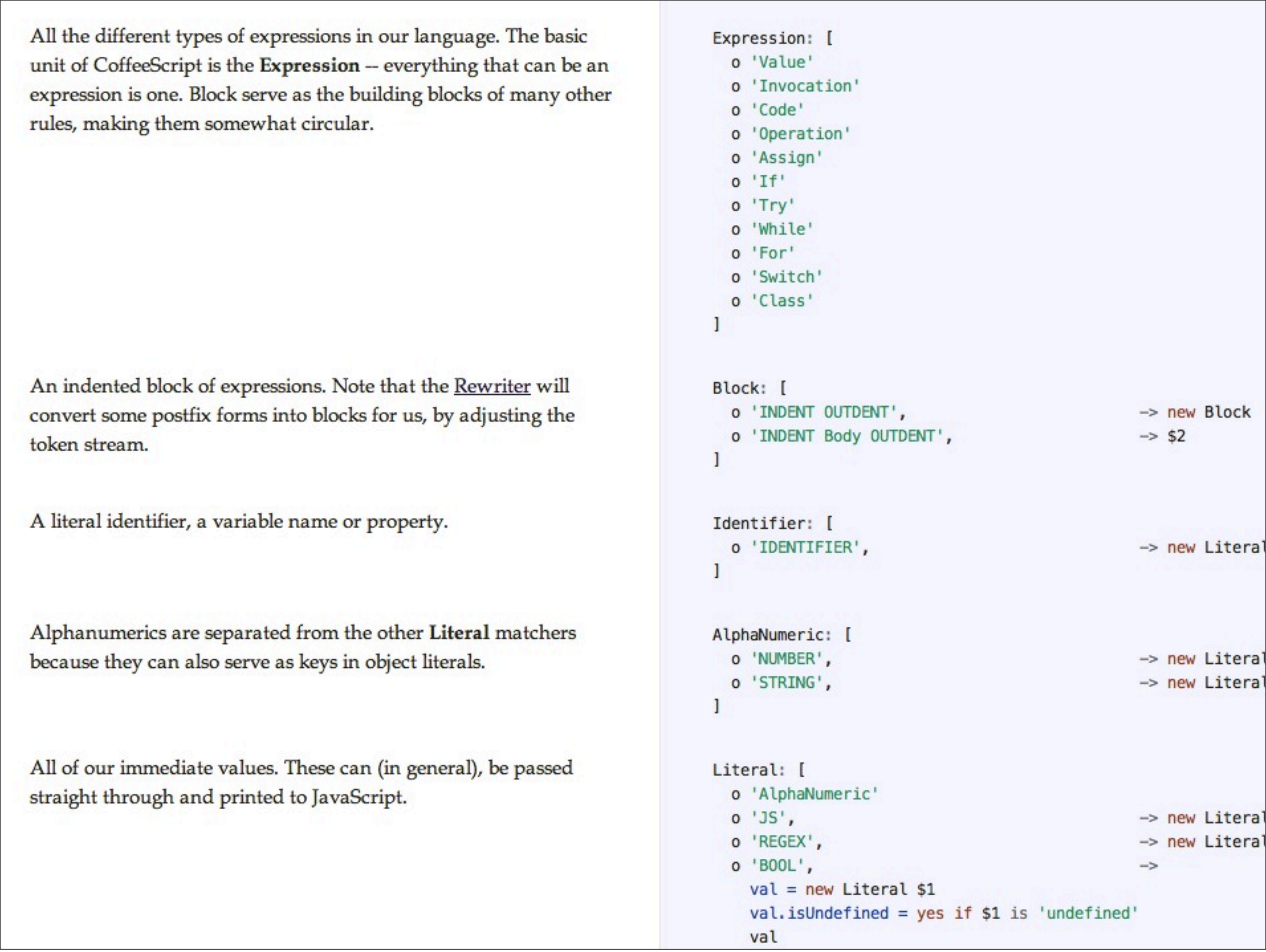
```
# Significant Whitespace

if condition
  do action

if condition then do action

do action if condition


# Other whitespace constructs, like try/catch.

try
  thing
catch error
  recover

try thing catch error then recover
```

```
# Bound vs. Unbound Functions.

class Book

  save: ->
    jQuery.ajax this.url, this.data, (response) ->
      merge this.data, response.data

# Why are unbound functions necessary in JavaScript?
# Executable Classes

class Pirate
  loot: ->
    say "Give me the gold!"


# ... and with if/else.

class Pirate
  if century > 1700
    loot: ->
      say "Give me the gold!"
  else
    loot: ->
      say "¡Dame el oro!"


# ... and with wrapped functions.

class Pirate
  loot: heartily ->
    say "Give me the gold!"
```

```
# Comprehensions (a little bit of everything)

for item in list
  process item

for key, value of object
  process value


# Own keys...

for own key, value of object
  process value


# Filtering keys and values.

for num in list when num % 2 is 0
  even num


# Value of a comprehension?
```

B.Y.O.J.

Build Your Own JavaScript



All the different types of expressions in our language. The basic unit of CoffeeScript is the **Expression** – everything that can be an expression is one. Block serve as the building blocks of many other rules, making them somewhat circular.

```
Expression: [
  o 'Value'
  o 'Invocation'
  o 'Code'
  o 'Operation'
  o 'Assign'
  o 'If'
  o 'Try'
  o 'While'
  o 'For'
  o 'Switch'
  o 'Class'
]
```

An indented block of expressions. Note that the Rewriter will convert some postfix forms into blocks for us, by adjusting the token stream.

```
Block: [
  o 'INDENT OUTDENT',                                   -> new Block
  o 'INDENT Body OUTDENT',                              -> $2
]
```

A literal identifier, a variable name or property.

```
Identifier: [
  o 'IDENTIFIER',                                       -> new Litera
]
```

Alphanumerics are separated from the other **Literal** matchers because they can also serve as keys in object literals.

```
AlphaNumeric: [
  o 'NUMBER',                                           -> new Litera
  o 'STRING',                                           -> new Litera
]
```

All of our immediate values. These can (in general), be passed straight through and printed to JavaScript.

```
Literal: [
  o 'AlphaNumeric'
  o 'JS',                                               -> new Litera
  o 'REGEX',                                            -> new Litera
  o 'BOOL',                                             ->
    val = new Literal $1
    val.isUndefined = yes if $1 is 'undefined'
    val
```

It's OK to **Cheat**.

Jison

PEG.js

Parser Generator for JavaScript

p4js

ReParse

OMeta

Canopy

Cruiser.Parse

JS/CC

Antlr

jsparse

JSGLR

});