CoffeeScript

CoffeeScript

The little language to make JS better

Piotr Klibert, Paweł Romanowski

April 10, 2012

https://github.com/promanow/coffeescript-slides/

What is it about?

What is JS?

- higher order functions
- lexically scoped lambdas
- dynamic typing, dynamic context (**this**)
- object model based on prototypes (BTW: Io language, also Self, LPC)
- variable number of function arguments
- callbacks, events, asynchronicity

What is JS?

- higher order functions
- lexically scoped lambdas
- dynamic typing, dynamic context (**this**)
- object model based on prototypes (BTW: Io language, also Self, LPC)
- variable number of function arguments
- callbacks, events, asynchronicity

- in short: modern, cool higher-level programming language
- . . . used widely

Really, why does it suck?

- C-like, verbose and bloated syntax
- `with` statement, fallthrough `switch`
- broken exception handling (can't specify which exceptions)
- differences between implementations
- `typeof` operator
- painful debugging
- semicolor insertion
- lack of dictionaries (maps, associative arrays)

Really, why does it suck? (pt. 2)

- no modules, implicit global scope of variables
- Java-like distinction between objects and primitives
- new block $\neq$ new scope

Really, why does it suck? (pt. 2)

- no modules, implicit global scope of variables
- Java-like distinction between objects and primitives
- new block $\neq$ new scope
- **type coercion rules**
  - == operator
  - counter-logic casts and promoting rules
  - `===` / `!==` syntax is just stupid
  - `http://wtfjs.com` if you're not convinced yet
- and so on . . .

Attempts to make JS better

- libraries
  - general: Underscore.js, Functional.js, . . .
  - DOM related: jQuery, MooTools, Prototype.js, Dojo, . . .
  - OOP related: Class.js, Base.js, Backbone.js, . . .

Attempts to make JS better

- libraries
  - general: Underscore.js, Functional.js, . . .
  - DOM related: jQuery, MooTools, Prototype.js, Dojo, . . .
  - OOP related: Class.js, Base.js, Backbone.js, . . .
- they all are quite good
- but we'd like some more
- it's the core syntax that fails

Enter CoffeeScript

- an attempt to make working with JS pleasant
- WITHOUT changing language semantics (GWT, Objective-J, others)
- Python- and Ruby-flavored syntax over C-flavor
- motto: “It’s just JavaScript!” - readable and linted at that
- encourages “best practices” of JS
- written itself in CoffeeScript
- can compile CS$\rightarrow$JS in the browser on the fly

Prerequisites

- **understanding and respect for JS semantics**
- conviction that readability counts
- belief that syntax matters
- courage to try sth new

Integration

- works extremely well with Underscore and Backbone
- in fact, works well with every JS library in existence
- node + coffee for quickly trying some code
- fast compilation
- good support for CS in IDEs (Vim, ofc, rulez!)
- out-of-the-box, transparent support in Django-pipeline

(Quick) syntax tutorial: basics

Note

For the sake of debugging, it's important to know what your code compiles to.

```
# functions
simpleFunc = ->
    5   # implicit return

# works similar to *args in Python
func = (args...) ->
    console.log arg for arg in args
    null

# can call functions without braces
func "some", "args", "for", "func"
func ["some", "args", "for", "func"]... # unpack
```

Syntax tutorial: default arguments

```
pow = (x, p = 2) ->
    ret = 1
    for i in [0..p]  # range
        ret *= x
    ret

pow 9  # 9 squared
pow(9, 3)  # 9 cubed
```

Syntax tutorial: strings

```
s = "Multiline strings work
        as in Python!" # indents converted to spaces

s2 = """A string without indents
        stripped"""    # will contain \n etc

# string interpolation
templ = (ctx) ->
    "I would like some #{ctx.what} please!"

# call templ
msg = templ({what: "Coffee"})

# CoffeeScript's fancy way
msg = templ
    what: "Coffee"
```

More on strings and function calling

```
templ = (ctx) ->
    "I would like some #{ctx.what} please!"

console.log templ {what: "Coffee"}
console.log templ
    what: "Coffee"

# These are all valid:
console.log templ({what: "Coffee"})
console.log (templ {what: "Coffee"})
console.log(templ {what:"Coffee"})

# there could be anything in #{}
"counting: #{x for x in [1..10]}"
"side effect of interpolation: #{console.log [1..10]}"
```

Iteration, comprehensions

```
numbers = (x for x in [1..10])  # [1, ..., 10]
# every loop returns a list of values from each step
numbers = for x in [1..10]
    x

numbers = [1..10]  # inclusive
numbers2 = [1...10]  # exclusive

# slices
slice = numbers[3...5] # equivalent of Python's [3:5]

# destructuring assignment (should be in the future JS)
[first, middle..., last] = [1..10]

first == 1 and last == 10 and middle.length == 8
```

Iteration, comprehensions, pt.2

```coffeescript
obj =
    property: "value"
    number: 9

for k, v of obj  # note the "of"!
    console.log k, v

# keys only
for k of obj
    console.log k

# iteration through object's own properties
# WITHOUT inherited ones (instead of hasOwnProperty)
for own k, v of otherObj
    console.log k, v
```

Classes, @ = this

Note

Have you read Backbone.js sources? You really should - `class` and `extends` keywords implement the same inheritance model that `Backbone.View.extends` provides.

```
class C
    # default value with assignment
    constructor: (@x = 42) ->    # empty func. body
    getX: ->
        @x

c = new C
c.getX()      # 42, needs ()!

cc = new C 55
cc.x      # 55
```

Classes, inheritance

```
class Base
    constructor: (args...) ->
        @args = args

class Sub extends Base
    constructor: (args...) ->
        # :: stands for .prototype
        args = (String::toUpperCase.call(x) for x in args)
        super(args...)  # unpack args

sub = new Sub("some", "silly", "strings")

sub instanceof Sub  # true
sub instanceof Base  # also true! for free!
```

Classes: @ binding via => (and Backbone integration)

```
class MyView extends Backbone.View
    events:
        "click .my-button" : "buttonClicked"

    initialize: () ->
        # => arrow defines a function that
        # binds @ (this)
        @el.find(".second-button").bind 'click', (ev) =>
            @secondClicked = true

    buttonClicked: (ev) =>
        # automatically bound to each instance
        @buttonClicked = true
```

Other syntax sugar

```
# chained comparisons
a = if 0 < i < 10 then i else "Not in range!"

# existential operator and assignment
if not variable?
    variable = "default value"

# shorter form
variable ?= "default value"

# also in place of accessor '.'
obj = {property: "value"}
obj.property.toUpperCase()  # ok
obj.otherProperty.toUpperCase()  # TypeError!
obj.otherProperty?.toUpperCase()  # undefined
```

Other syntax sugar pt.2

```
# fixed "switch"
switch command
    when "work" then doWork()
    when "chill", "relax"
        goToHawaii "now"
    else doNothing()

# using destructuring to swap
x = -666
y = 0

[x, y] = [y, x]
```

Other things worth mentioning

- CoffeeScript encloses all output code into anonymous JS function
  - enable `-b` compiler switch to disable this behavior
  - to create global objects:

    ```
    window.globalObject = 5
    # or use pattern
    root = this
    ```

- You can try Coffeescript online at http://coffeescript.org/

Example JS output

```
# Just a trivial example
# More at http://coffeescript.org

class C
    constructor: (@x = 42) ->



    getX: ->
        @x



c = new C
c.getX()
```

Example JS output

```coffeescript
# Just a trivial example
# More at http://coffeescript.org

class C
    constructor: (@x = 42) ->



    getX: ->
        @x




c = new C
c.getX()
```

```javascript
(function() {
var C, c;

C = (function() {
    function C(x) {
    this.x = x != null ? x : 42;
    }


    C.prototype.getX = function() {
    return this.x;
    };
    return C;
})();
c = new C;
c.getX();
}).call(this);
```

Summary

Let's sum up.

Things CoffeeScript does well

1. it's just JS with terser syntax
2. fixes some broken JS syntax
3. typically $\frac{1}{3}$ less code to write
4. exposes "good parts" of JS
5. hides "bad parts" at the same time
6. no *runtime* performance penalty
7. integrates well
8. stable, production-ready
9. fun to write, appeals to Pythonistas ;)

What's not so cool?

1. debugging - we debug result JS
2. not a separate language
   - requires JS understanding
   - won't fix everything
   - sometimes strange things may happen (it's JS after all)
3. does not solve some JS problems
   - modules, namespaces
   - you still need to handle that yourself
4. (?) is not as mature and rock-solid as JS itself

Further reading

- Official site - learn by example + side-by-side comparisons
  http://coffeescript.org/
- The Little Book on CoffeeScript
  http://arcturo.github.com/library/coffeescript
- CoffeeScript, Meet Backbone.js: A Tutorial
  http://adamjspooner.github.com/coffeescript-meet-backbonejs/

Other Coffee-based languages to check out

- coco - CoffeeScript meets Perl and Haskell
  https://github.com/satyr/coco
- CoffeeKup - CS based HTML markup
  http://coffeekup.org/
- ...

End

Thanks for your attention.

End

Thanks for your attention.

Questions?