Functional programming in JavaScript ecosystem

@paulmillr

JS is a functional language

JS is a functional language

Sort of...

What do we have today

What do we have today

Proper anonymous functions (λ)

Closures

ES5 array extras (map, filter, reduce...)

Is it enough?

Is it enough?

Yep.

Is it comfy?

Is it comfy?

Nope.

What's wrong?

What's wrong?

keywords

are too long

What's wrong?

keywords

are too long

braces

everywhere

What's wrong?

keywords

are too long

braces

everywhere

no **static**

types

What's wrong?

keywords

are too long

braces

everywhere

no **static**

types

no proper

tail calls

What's wrong?

keywords

are too long

braces

everywhere

this scoping problems

no **static**

types

no proper

tail calls

What's wrong?

keywords

are too long

no constants

braces

everywhere

this scoping problems

no static

types

no proper

tail calls

What's wrong?

ES5 array extras
work alright in **chaining**
But

Prototype-based
== awful modularity
== collisions
== low performance

Solutions?

I want to write functionally in JS ecosystem simply. What are my options?

Solutions?

haskell-to-js ClojureScript

Solutions?

haskell-to-js ClojureScript

Terrible

interoperability

Compile to
very long files
hard to debug

Solutions?

Readable / reasonable JS output?

Good interoperability?

Simple to debug?

CoffeeScript

coffeescript.org

CoffeeScript

Great small language

Compiles down to JS

#11 most used on GitHub

Used in 1000s of popular projects

CoffeeScript

Better for functional programming

Heals JS quirks

CoffeeScript

Implicit return

Short λ declaration

Whitespace-significant syntax

```
(a, b, c) -> a * b / c
```

vs

```
function(a, b, c) {
  return a * b / c;
}
```

CoffeeScript

No curly braces

Round braces are optional

```
times(2, sum(1, 2, 3))  # => 12

times 2, sum 1, 2, 3  # => 12
```

CoffeeScript

List comprehensions

```
(a * 2 for a in [10,20,40])
```

CoffeeScript

this fixes via bound functions

```
current = this
fn = =>
  log current == this
$('body').on 'click', fn
# Will log true
```

```
var current = this;
var fn = function() {
  log current == this;
};
$('body').on 'click', fn
# Will log false
```

CoffeeScript

Doesn't heal all quirks

Brings own ones

CoffeeScript

Chaining is a lot readable with short λs, but still terrible

```
# Doesn’t work on Array-like objects
document.querySelectorAll(‘.user’)
  .map((x) -> x + 5)
  .maximum()
# Defining methods on prototypes? No, thanks.
```

CoffeeScript

Must create λs even for simple stuff

```
array
  .map((a)  => a + 2)
  .filter((a) => a != 10)
  .reduce((a, b) => Math.min(a, b))
```

the only real work

CoffeeScript

List comprehensions aren't real

Basically an infix **for** loop

```
(a * b for a in [1, 2, 3] for b in [10, 20, 40])
# non flattened result, order is wrong
# => [ [ 10, 20, 30 ], [ 20, 40, 60 ], [ 40, 80, 120 ] ]
```

CoffeeScript

Terrible variable scoping

```
variable = 1
fn = ->
  variable = 2
fn()
console.log variable  # => 2
```

Roy

roy.brianmckenna.org

Roy

Type inference

Algebraic data types

Pattern matching

Monadic syntax

Roy

Not ready yet

Still a lot of stuff
it doesn't have

LiveScript

LiveScript

gkz.github.com/LiveScript/

LiveScript

JS +  = 

 +  = Coco

Coco +  = LiveScript

LiveScript

Easy transition from Coffee

Improved **readability**

Perfect **piping** operators

|> (F#)

<| (F#) ($ in Haskell)

LiveScript

Standard library (prelude.ls)

Inspired by prelude.hs

gkz.github.com/prelude-ls/

LiveScript

Partially applied operators and member access

```
array
  |> map (+ 2)
  |> filter (!= 10)
  |> maximum
```

LiveScript

Compile-time consants

Also, compiler flag that make all vars consts

```
const string = ‘hello’
string = 5710
# => Error
```

LiveScript

Improved var scoping

```
a = 1
do ->
  a = 2
a # => still 1
```

LiveScript

Improved operators associativity

unique pulls .length
unique node or not empty node

instead of coffee's

(unique pulls).length
(unique node) or not (empty node)

LiveScript

Real list comprehensions

```
[x ** y for x in [10,20] for y in [2,3]]
# => [100,1000,400,800]
```

LiveScript

Pattern matching

```
take(n, [x, ...xs]:list) =
  | n <= 0           => []
  | empty list  => []
  | otherwise  => [x] +++ take n - 1, xs
```

LiveScript

Simple currying

```
times = (x, y) --> x * y
times 2, 3          # => 6
double = times 2
double 5            # => 10
```

LiveScript

Async callback flattening syntax

```
error <- fs.write-file path, data
```

LiveScript

Is it ready to use today?

LiveScript

Is it ready to use today?

Yep!

1.0.0 will be released later this week.

LiveScript

Debugging

Relatively simple

Will be super simple with source maps (2012)

LiveScript

HTML5 apps

Sure!

Including builders that auto-compile your apps without headache (Brunch.io).

LiveScript

Node.js

Yep.

Just add pre-publish
hook to `package.json`

Compare

Compare

LiveScript (w/prelude)

```
users
 |> map (.age)
 |> filter (> 10)
 |> maximum
```

Coffee

```
users
 .map((u) -> u.age)
 .filter((a) -> a > 10)
 .reduce (a, b) ->
  Math.max a, b
```

JS

```
users
 .map(function(u) {return u.age})
 .filter(function(a) {return a > 10})
 .reduce(function(a, b) {
  return Math.max(a, b)
 });
```

Compare

LiveScript

```
elems = document.query-selector-all '.listing .meta a:nth-child(3)'
pulls = elems |> map (.inner-text)
text  = "Total #{pulls.length} pull requests in #{unique pulls .length} repos."
```

Coffee

```
elems = [].slice.call document.querySelectorAll '.listing .meta a:nth-child(3)'
pulls = elems.map (elem) -> elem.innerText
unique = elems.reduce (a, b) ->
  a.push(b) if b not in a
  a
text  = "Total #{pulls.length} pull requests in #{(unique pulls).length} repos."
```

JS → 14 LOC

Compare

LiveScript

```
quick-sort = ([x, ...xs]:list) ->
  | empty list  => []
  | otherwise  =>
    [left, right] = partition (<= x), xs
    (quick-sort left) +++ [x] +++ (quick-sort right)
```

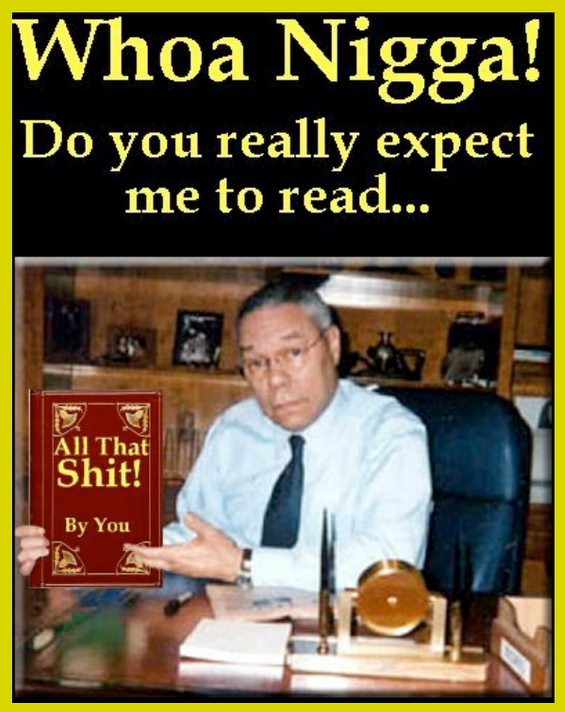
gist.github.com/3074009

Compare

LiveScript

```
quick-sort = ([x, ...xs]:list) ->
  | empty list  => []
  | otherwise  =>
    [left, right] = partition (<= x), xs
    (quick-sort left) +++ [x] +++ (quick-sort right)
```

gist.github.com/3074009



Coffee

JS

Future

ECMAScript 6

CoffeeScript 2.0

LiveScript.next

Future: ECMAScript 6

New javascript standard

let block-scoped vars

const value checking

Short **arrow** functions

Tail call optimization

Real **list comprehensions**

Future: ECMAScript 6

Still a lot of syntax garbage

```
((a, b) => {a + b})(2, 5));
```

vs

```
(+) 2, 5
```

Future: CoffeeScript 2

Same feature set

Proper compiler design principles

github.com/michaelficarra/CoffeeScriptRedux

Future: CoffeeScript 2

When it will be ready,
author will create
a functional fork

github.com/michaelficarra/
coffee-of-my-dreams

Future: LiveScript

Type inference

Pure annotations

Tail call optimization

So?

I want to write functionally in JS ecosystem simply.

What are my options?

1. Use LiveScript
2. Wait for fork of Coffee 2.0
3. Wait for Roy

Thanks!

Paul Miller

paulmillr.com

@paulmillr