CoffeeScript: The Good Parts

The good and not so good parts of CoffeeScript in comparison to JavaScript

@azat_co

About The Presenter

Azat Mardan

- Worked in the US government, startups and corporations
- Wrote six books on JavaScript and Node.js (the latest is *Practical Node.js* by Apress)
- Certified yoga teacher and paleo lifestyle enthusiast

Fun story: “CoffeeScript is a solution without a problem”

I used to make fun of CoffeeScript, before falling in love with it.

Who likes /uses
CoffeeScript already?

Please be open-minded

Blub Paradox

Blub paradox: devs are satisfied with whatever language they happen to use, because it dictates the way they think about programs.

http://en.wikipedia.org/wiki/Paul_Graham_(computer_programmer)#Blub

CoffeeScript's Bad Rep

Most of the people who say bad things about CoffeeScript have **never** built with it anything relatively large-scale and production-ready.

CoffeeScript at DocuSign

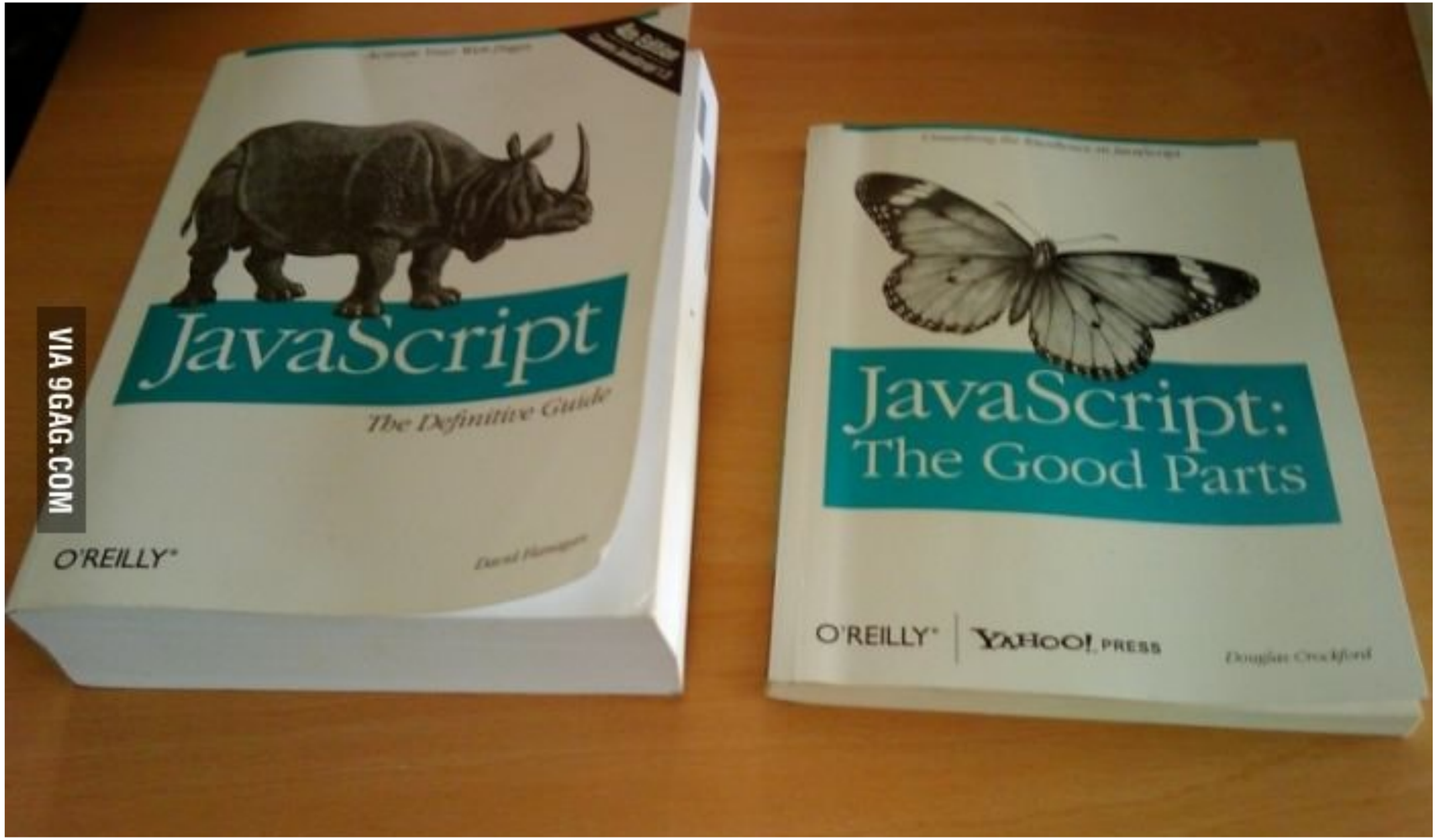
DocuSign stack: CoffeeScript+Node.js+Backbone.js+Grunt

Observation: front-end developers, after only **a few weeks** of CoffeeScript, **didn’t want** to go back to regular JavaScript!

“CoffeeScript is a little language that compiles into JavaScript.” — coffeescript.org

TOC for v1.7.1

Maybe CoffeeScript is not so small anymore?


VIA 9GAG.COM
JavaScript
The Definitive Guide
O'REILLY
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS

JavaScript

Native JavaScript Issues

- == vs ===
- Functional inheritance vs pseudo-classical
- Global variable leakage (missing var)
- Many others

CoffeeScript: The Good Parts (some of them)

- Syntax: curly braces, parentheses and semicolons
- Function declaration: -> and =>
- Scoping: automatic var
- Iterators / comprehensions: replacement of for loops
- Class declaration: class operand

Syntax

- No semicolons, i.e., they are banned
- Logical blocks can omit curly braces
- Function calls can omit parentheses

Why Semicolons are Bad? (*IMHO)

- Extra time&work to put them
- If missed, inconsistent but working code
- Redundant/not-needed (expect in two cases)
- Semicolon Insertion (ASI)

Logical Blocks (optional curly braces)

```
kids =
  brother:
    name: "Max"
    age:  11
  sister:
    name: "Ida"
    age:  9
```

Function calls

```
console.log object.class

$('.account').attr class: ‘active’

$(‘button’).css
  color: red
  “font-size”: “16px”
```

Function Declaration

- Skinny arrow (->) saves time typing “function” over and over again
- Fat arrow (=>), i.e., no need to use “var that = this” or “var self = this”

Function Declaration Elegance

CoffeeScript:

```
a = (x,y) -> console.log x+y
```

JavaScript:

```
var a;

a = function(x, y) {
  return console.log(x + y);
};

a(10, -5);
```

Function Declaration: Skinny Arrow

CoffeeScript:

```
console.log @
$('div').click ()->
  console.log @
```

JavaScript:

```
console.log(this);
$('div').click(function() {
  return console.log(this);
});
```

Prints “window” and DOM elements i.e.,
“this” changes and @ changes too

Function Declaration: Fat Arrow

CoffeeScript:

```
console.log @
$('div').click ()=>
  console.log @
```

JavaScript:

```
var _this = this;

console.log(this);

$('div').click(function() {
  return console.log(_this);
});
```

Prints “window” both times (i.e., the outer scope)

Scoping

- Manual “var” is banned
- Variables declared in the scope in which they are encountered first (i.e., the order in which variables used determines their scope).

Why auto vars are good? Missed “var” Example.

CoffeeScript:

```
var a = function (c) {
  b = 10;
  return b + c;
}
console.log(a(0));
```

b is window.b — bad!

JavaScript:

```
var a = function(c) {
  var b = 10;
  return b + c;
};
console.log(a(0));
```

b is a private attribute —

what we wanted!

Scoping: Example I

CoffeeScript:

```
b = 1
a = ->
  b = -1

a()
console.log b
```

JavaScript:

```
var a, b;
b = 1;
a = function() {
  return b = -1;
};
a();
console.log(b);
```

Scoping: Example II

CoffeeScript:

```
a = ->
  b = -1
b = 1
a()
console.log b
```

JavaScript:

```
var a, b;
a = function() {
  var b;
  return b = -1;
};
b = 1;
a();
console.log(b);
```

Iterators / Comprehensions (for loops)

Awesome time savers! Good for arrays and objects:

```
for item, index in arrayObject
  iterator(item)

for key, value of object
  iterator(item)
```

Iterating over an Array

CoffeeScript:

```
for item, index in arrayObject
  iterator(item)
```

JavaScript:

```
var index, item, _i, _len;

for (index = _i = 0,
  _len = arrayObject.length;
  _i < _len;
  index = ++_i) {
    item = arrayObject[index];
    iterator(item);
}
```

Iterating over an Object

CoffeeScript:

```
for key, value of object
  iterator(value)
```

JavaScript:

```
var key, value;

for (key in object) {
  value = object[key];
  iterator(value);
}
```

Iterators / Comprehensions (for loops) II

Many options:

```
iterator (item) for item in arrayObject

iterator item for item in arrayObject

iterator item for item in arrayObject when item > 0
```

Class Declaration

- In JavaScript classes are absent at all!
- CoffeeScript eloquently implements Pseudo-classical inheritance pattern: “new” and capitalized name (“new Animal”, “new Vehicle”, etc.)
- Pseudo-classical is hard and cumbersome without CS
- CoffeeScript “constructor” method and “super” call

Class Example

CoffeeScript:

```
class Vehicle
  constructor: (@name) ->
  move: (meters) ->
    console.log @name + " moved #{meters} miles."

camry = new Vehicle "Camry"
camry.move(50)
```

Output: Camry moved 50 miles.

Class Example

JavaScript:

```
var Vehicle, camry;

Vehicle = (function() {
  function Vehicle(name) {
    this.name = name;
  }

  Vehicle.prototype.move = function(meters) {
    return console.log(this.name + (" moved " + meters + " miles."));
  };

  return Vehicle;

})();

camry = new Vehicle("Camry");

camry.move(50);
```

2x: 6 vs 12 lines of code!

Other CoffeeScript Goodies

- Splats (e.g., “options…”)
- Conditions (if, isnt, not, and, or, unless)
- Arrays and their slicing (arr = [1..10], slicedArr = arr[2..4])

CoffeeScript: The Good Parts Summary

- Syntax: curly braces, parentheses and semicolons
- Function Declaration: -> and =>
- Scoping: automatic var
- Iterators / Comprehensions: replacement of for loops
- Class Declaration: class operand

CoffeeScript: Not So Good Parts

- Learning: 1-2 day, free online ebooks
- Compilation: extra build step (use Grunt or similar)
- Parentheses: optional and cause misinterpretation
- Debugging: use source-maps

CoffeeScript of My Dreams

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

How to Get Started

```
$ npm install -g coffee-script
$ coffee
>…
```

Companies that use CoffeeScript

- GitHub
- Dropbox
- DocuSign
- Airbnb (mobile)
- HotelTonight
- Basecamp (mobile)

Further Reading

CoffeeScript FUNdamentals: The Better JavaScript

http://bit.ly/1nD4dE3

CoffeeScript Ebooks

- The Little Book on CoffeeScript (free onine)
- CoffeeScript Cookbook (free online)
- Smooth CoffeeScript (free online)
- CoffeeScript Ristretto (free online)

Future / Alternatives

- Dart (early stage)
- TypeScript: MicroSoft project
- ECMAScript 6: be careful with old browsers, use shims, fully available after June 2015
- Sweet.js: macros!

Conclusions

- Good for enterprise and **large team,** because it's easer to have common style, e.g., https://github.com/styleguide/javascript
- Good for **developers new to JavaScript** and those coming from **OOP** languages (Java, Ruby)
- Cross-browser / old browser support
- More productive and happier developers (after learning)
- Tested, robust, and **available now**
- Awesome with **Backbone.js** and **Underscore.js**

Session Summary

- Native JavaScript Issues
- CoffeeScript: The Good Parts
- How to Get Started
- Future / Alternatives
- Conclusions

Discussion and Q&A Time

Questions, thoughts and experiences?