# Introduction to CoffeeScript

Michael Heraly

November 13, 2014

Presentation of JavaScript and CoffeeScript

Why CoffeeScript

Beginners' mistakes in CoffeeScript

CoffeeScript reference

Web browsers can only interpret JavaScript.

Web browsers can only interpret JavaScript.

But the problem is that JavaScript is ugly (and verbose).

Web browsers can only interpret JavaScript.

But the problem is that JavaScript is ugly (and verbose).

```
var cubes = (function() {
    var _i, _len, _results;
    _results = [];
    for (_i = 0, _len = list.length; _i < _len; _i++) {
        num = list[_i];
        _results.push(math.cube(num));
    }
    return _results;
})();</pre>
```

Web browsers can only interpret JavaScript.

But the problem is that JavaScript is ugly (and verbose).

```
var cubes = (function() {
    var _i, _len, _results;
    _results = [];
    for (_i = 0, _len = list.length; _i < _len; _i++) {
        num = list[_i];
        _results.push(math.cube(num));
    }
    return _results;
})();</pre>
```

While the equivalent code in CoffeeScript is:

Web browsers can only interpret JavaScript.

But the problem is that JavaScript is ugly (and verbose).

```
var cubes = (function() {
    var _i, _len, _results;
    _results = [];
    for (_i = 0, _len = list.length; _i < _len; _i++) {
        num = list[_i];
        _results.push(math.cube(num));
    }
    return _results;
})();</pre>
```

While the equivalent code in CoffeeScript is:

```
cubes = (math.cube(num) for num in list)
```

### Ugly JavaScript classes

If you need more complex JS, like classes, here is the code:

#### Ugly JavaScript classes

If you need more complex JS, like classes, here is the code:

```
var Animal = (function() {
  function Animal(name) {
    this .name = name;
 Animal.prototype.move = function(meters) {
    return alert(this.name + (" moved " + meters + "m."));
  };
 return Animal;
})();
```

#### Ugly JavaScript classes

If you need more complex JS, like classes, here is the code:

```
var Animal = (function() {
  function Animal(name) {
    this .name = name:
  Animal.prototype.move = function(meters) {
    return alert(this.name + (" moved " + meters + "m."));
  };
 return Animal;
})();
```

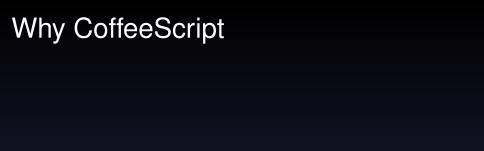
...And there is still no inheritance involved...

#### And CoffeeScript classes

The CoffeeScript syntax is closer to the Python one.

```
class Animal
constructor: (@name) ->

move: (meters) ->
alert @name + " moved #{meters | m. "
```



Easier than JavaScript

- Easier than JavaScript
- Exposes the good parts of JS (and avoid the bad ones)

- Easier than JavaScript
- Exposes the good parts of JS (and avoid the bad ones)
- Cleaner code.

- Easier than JavaScript
- Exposes the good parts of JS (and avoid the bad ones)
- · Cleaner code.
- Write less, do more.

CoffeeScript is translated into JavaScript to allow web browsers to interpret the code.

CoffeeScript is translated into JavaScript to allow web browsers to interpret the code.

So...why not directly write JavaScript?

CoffeeScript is translated into JavaScript to allow web browsers to interpret the code.

So...why not directly write JavaScript?

As shown previously, CoffeeScript is really better to keep a clean and maintainable code.

It is a kind of great syntactic sugar, so you can still use libraries like jQuery.

CoffeeScript is translated into JavaScript to allow web browsers to interpret the code.

So...why not directly write JavaScript?

As shown previously, CoffeeScript is really better to keep a clean and maintainable code.

It is a kind of great syntactic sugar, so you can still use libraries like jQuery.

If you're still not convinced, check out the differences of code you need to write!

### CoffeeScript syntax differences with JavaScript

- No keyword var to declare a variable before using it.
- No { } to delimit the if-else blocks.
- () to surround a condition in if are not necessary.
- · No; at the end of instructions.
- Declare a function square with a parameter x :

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

- Returns are implicit.
- Use and instead of &&.
- Use or instead of II.
- Use # to put inline comments
   (and ###...### is equivalent to /\* ...\*/)

## Beginners' mistakes to avoid in CoffeeScript

Like Python, indentation is made by (at least) 2 spaces.

It is the only way to define a code block (in a method, if, else, ...).

## Beginners' mistakes to avoid in CoffeeScript

Example of a "static" CoffeeScript class:

```
class window. Utils
@calc: (x, y) ->
...
```

- class window. Utils attaches the Utils class to the window object (containing the DOM document).
- The @ character declares that the calc method is static.
- x and y are the parameters of the calc method.
- If you call a static method from another class, the syntax is <Class name>.<method name>.
- If you call a static method from the same class, you can use the syntax @<method name>.

## Beginners' mistakes to avoid in CoffeeScript

Example of a CoffeeScript class:

```
class window.Person:
@name

constructor: (name) ->
@name = name

getName: () =>
@name
```

- The @ means that name is an instance field.
- The return of the getName method is implicit.
- As getName is an instance method, => must be used (instead of ->) to define that this relates to the corresponding Person object, and not to the this where the method has been called. (that is due to one of the quirks of JS)

#### In case you really need it

If you want to use something that doesn't exist in CoffeeScript (shouldn't happen), you can embed JS code between '...':

```
hi = 'function() {
  return [document.title, "Hello JavaScript"].join(": ");
}'
```

But use this only if you need it.

It is much better to keep a coherent code in CoffeeScript.

#### For more informations

CoffeeScript reference documentation:

http://coffeescript.org