@zachdennis

This is CoffeeScript.

Mutually human

CoffeeScript

```
number   = 42
opposite = true

number = -42 if opposite

square = (x) -> x * x

list = [1, 2, 3, 4, 5]

math =
  root:   Math.sqrt
  square: square
  cube:   (x) -> x * square x

race = (winner, runners...) ->
  print winner, runners

alert "I knew it!" if elvis?
```

7 things to know

1. Compiles into JavaScript.

```coffeescript
outer = 1

changeNumbers = ->
  inner = -1
  outer = 10

inner = changeNumbers()
```

```javascript
var changeNumbers, inner, outer;

outer = 1;

changeNumbers = function() {
  var inner;
  inner = -1;
  return outer = 10;
};

inner = changeNumbers();
```

CoffeeScript

JavaScript

2. White-space Sensitive

```coffeescript
changeNumbers = ->
  inner = -1
  outer = 10
```

```javascript
changeNumbers = function() {
  var inner;
  inner = -1;
  return outer = 10;
};
```

```coffeescript
badChangeNumbers = ->
inner = -1
outer = 10
```

```javascript
badChangeNumbers = function() {};
inner = -1;
outer = 10;
```

CoffeeScript

JavaScript

3. Use existing JavaScript

```
clicked = ->
  console.log "clicked"

jQuery ->
  $("button").on "click", clicked
```

CoffeeScript

4. Use existing CoffeeScript

```
# clicked.coffee
clicked = ->
  console.log "clicked"
```

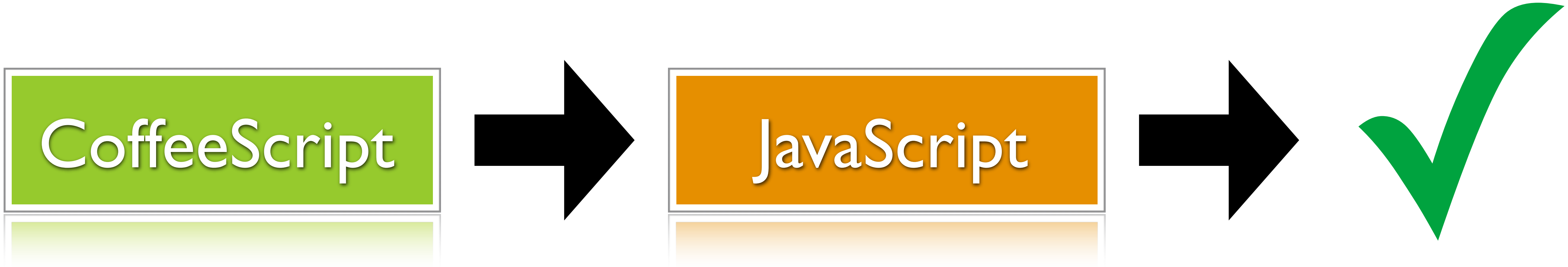
CoffeeScript

```
// app.js
jQuery(function() {
  $("button").on ("click", clicked);
});

clicked();
```

JavaScript

5. Generated JavaScript passes JSLint



6. Play with CoffeeScript Online

http://www.coffeescript.org > Try CoffeeScript

http://js2coffee.org > Coffee → JS

7.

It’s JavaScript without the unnecessary syntax

more thing
before we
begin

1. Compile your coffee

```
> coffee -c foo.coffee
```

```
> coffee -co javascripts/ source/
```

33

language features
approx. minutes

1. Variable Assignment

You never need to write *var* yourself.

```
outer = 1

changeNumbers = ->
  inner = -1
  outer = 10
```

```
var changeNumbers, inner, outer;

outer = 1;

changeNumbers = function() {
  var inner;
  inner = -1;
  return outer = 10;
};
```

CoffeeScript

JavaScript

2. Conditional Assignment

```
value = computeBigNumber()
value ||= computeBigNumber()
```

CoffeeScript

```
var value;

value = computeBigNumber();

value || (value = computeBigNumber())
```

JavaScript

3. Traditional conditionals

Remove *unnecessary* syntax.

```
if condition1
  doThing1()
else if condition2
  doThing2()
else
  doThing3()
```

CoffeeScript

```
if(condition1) {
  doThing1();
} else if (condition2) {
  doThing2();
} else {
  doThing3();
}
```

JavaScript

4. One line conditionals

Adding *readability*.

```
congratulate() if winner

showErrors() unless success
```

CoffeeScript

```
if(winner){
  congratulate();
}

if(!success){
  showErrors();
}
```

JavaScript

5. Conditional Operators

Safety first.

```
foo == bar
foo != bar
```

CoffeeScript

```
foo === bar
foo !== bar
```

JavaScript

6. Conditional Operators

Plain *english*.

```
foo and bar
foo or bar
```

```
foo && bar
foo || bar
```

CoffeeScript

JavaScript

7. Conditional Operators

Saying what you *mean*.

```
action is "clicked"
action isnt "clicked"
```

```
action === "clicked"
action !== "clicked"
```

CoffeeScript

JavaScript

8. Conditional Operators

More ways to say what you *mean*.

```
admin is true
admin is false

checked is yes
checked is no

value is on
value is off
```

CoffeeScript

```
admin === true
admin === false

checked === true
checked === false

value === true
value === false
```

JavaScript

9. Conditional Operators, In Array

Checking array *membership*.

```
odds = [1, 3, 5, 7, 9, 11]

if 1 in odds
  console.log "It’s odd!"
else
  console.log "It’s even!"
```

CoffeeScript

10. Conditional Existence

Checking *existence* of a variable/property.

```
car ?= {}
car.speed ?= 75
```

CoffeeScript

CoffeeScript's existential operator ? returns true unless a variable is null or undefined.

11. Conditional Existence cont...

What about ||= instead of ?=

```
car = {}
car.speed = 0
car.speed ||= 75
```

CoffeeScript

||= will overwrite a speed of 0 with 75 when it shouldn't. ?= is safer.

12. Arrays

Single line or *multi-line* array definitions.

```
fruits = ["apples", "bananas"]


fruits = [
  "apples"
  "bananas"
]
```

```
var fruits;

person = [
  "apples",
  "bananas"
];
```

CoffeeScript

JavaScript

13. Objects

Single line or *multi-line* object definitions.

```
person = name: "Joe", age: 39


person =
  name: "Joe",
  age: 39
```

```
var person;

person = {
  name: "Joe",
  age: 39
};
```

CoffeeScript

JavaScript

14. Functions

Parameter-less function definitions.

```
hello = ->
  "Zach"

hello = () ->
  "Zach"

hello()

# doesn’t call function
hello
```

CoffeeScript

```
var hello;

hello = function(){
  return "Zach";
};
```

JavaScript

15. Functions

Parameter-filled function definitions.

```
hello = (name) ->
 "Hello " + name

hello("Joe")

hello "Joe"
```

```
var hello;

hello = function(name){
  return "Hello " + name;
};
```

CoffeeScript

JavaScript

16. Functions

In-line function definitions.

```
$("button").on "click", -> alert("Hello " + name)


$("button").on "click", ->
  alert("Hello " + name)

$("button").on "click", (e) ->
  e.preventDefault()
  alert("Hello " + name)
```

CoffeeScript

JavaScript

16 and a half.

Everything is an *expression*.

```coffeescript
grade = (student) ->
  if student.excellentWork
    "A+"
  else if student.okayStuff
    if student.triedHard
      "B"
    else
      "B-"
  else
    "C"
```

CoffeeScript

```javascript
var grade;

grade = function(student) {
  if (student.excellentWork) {
    return "A+";
  } else if (student.okayStuff) {
    if (student.triedHard) {
      return "B";
    } else {
      return "B-";
    }
  } else {
    return "C";
  }
};
```

JavaScript

17. Splats

Variable-length parameter lists (aka *splats*).

```
hello = (names...) ->
 "Hello " + names

hello("Joe", "Tim", "Jim")

hello "Joe", "Tim", "Jim"
```

```
var hello,
  __slice = [].slice;

hello = function(){
  var names;
  names = 1 <= arguments.length ?
    __slice.call(arguments, 0) :
    [];
  return "Hello " + names;
};
```

CoffeeScript

JavaScript

18. String Interpolation

Double quotes for interpolation.

```
"Hello #{name}!"                 "Hello " + name + "!"
```

Single quotes for string literals.

```
'Hello #{name}!'                 "Hello #{name}"
```

CoffeeScript

JavaScript

19. Array comprehensions

Collecting values *over* elements.

```
numbers = [1, 2, 3, 4]

negatives = (-num for num in numbers)

# negatives = [-1, -2, -3, -4]
```

CoffeeScript

20. Array comprehensions

The *by* modifier.

```
numbers = [1, 2, 3, 4]

negatives = (-num for num in numbers by 2)

# negatives = [-1, -3]
```

CoffeeScript

21. Array comprehensions

The *when* modifier.

```
numbers = [1, 2, 3, 4]

negatives = (-num for num in numbers when num > 2)

# negatives = [-3, -4]
```

CoffeeScript

22. Object comprehensions

Over properties with *of*.

```
person = name: "Joe", age: 39

properties = (name for name of person)

# properties = ["name", "age"]
```

CoffeeScript

23. Object comprehensions

Over *properties* and *values* with *of*.

```
person = name: "Joe", age: 39

properties = ([name, value] for name, value of person)

# properties = [["name", "Joe"], ["age", 39]]
```

CoffeeScript

24. Ranges

```
[1..10]        [1,2,3,4,5,6,7,8,9,10]

[10..1]        [10,9,8,7,6,5,4,3,2,1]
```

CoffeeScript

JavaScript

25. Comments

Single line and *block* comments!

```
# This is a single line comment

###
This is a
block comment
###
```

CoffeeScript

27. De-structuring assignment

Array pattern matching.

```
[foo, bar, baz] = ["foo", "bar", "baz"]

# equivalent to:
foo = "foo"
bar = "bar"
baz = "baz"

# also equivalent to:
arr = ["foo", "bar", "baz"]
foo = arr[0]
bar = arr[1]
baz = arr[2]
```

CoffeeScript

28. De-structuring assignment

Swapping values.

```
foo = 1
bar = 2

[foo, bar] = [bar, foo]

# foo = 2
# bar = 1
```

CoffeeScript

29. De-structuring assignment

With *splats*.

```
numbers = [1, 2, 3, 4]

[head, tail...] = numbers

# head = 1
# tail = [2,3,4]
```

CoffeeScript

30. De-structuring assignment

With *objects*.

```
person = name: "Joe", age: 39

{name: myName, age: myAge} = person

# myName = “Joe”
# myAge = 39
```

CoffeeScript

31. De-structuring assignment

Shorthand when variable and property names are the same.

```
person = name: "Joe", age: 39

{name: name, age: age} = person

# the above is shorthand for
{name, age} = person

# name = “Joe”
# age = 39
```

CoffeeScript

32. this

@ is short-hand for *this*.

```
@
```

```
this
```

```
$("a").on "click", ->
  alert @.attr("href")
```

```
alert(this.attr("href"))
```

```
# you can omit the dot
$("a").on "click", ->
  alert @attr("href")
```

```
alert(this.attr("href"))
```

CoffeeScript

JavaScript

33. Classes

```coffeescript
class Animal

tim = new Animal "Tim the turtle"
```

CoffeeScript

33b. Classes

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

  # instance method
  move: (meters) ->
    alert "#{@name} moved #{meters}m."

tim = new Animal "Tim the turtle"
tim.move 10
```

CoffeeScript

33c. Classes

```
class Animal
  # class method
  @all: ->
    @animals ?= []

  constructor: (@name) ->
    Animal.all().push @

  # instance method
  move: (meters) ->
    alert "#{@name} moved #{meters}m."

tim = new Animal "Tim the turtle"
tim.move 10
Animal.all()    # [Animal instance]
Animal.animals # [Animal instance]
```

CoffeeScript

xx. Diving Deeper

Classes (super, inheritance, etc)
Namespaces.
Function bindings.
Block regular expressions.
Embedding JavaScript.
Exceptions.
Chained comparisons.

3 resources

Resources

CoffeeScript

http://www.coffeescript.org

http://coffeescript.org/#resources

Quick Ref Card

http://autotelicum.github.com/Smooth-CoffeeScript/CoffeeScript%20Quick%20Ref.pdf

Little Book on Coffee Script

http://arcturo.github.com/library/coffeescript/