Brewing Node Code w/ CoffeeScript

Node Hands On Meetup

10.17.12

Jeremy Smith
me@jeremyis.com, @jeremyis

CoffeeScript

- Compiles into JavaScript
- Adds minimal and clean syntactic sugar a la Ruby / Python / Haskell
- Has modern / cool language features

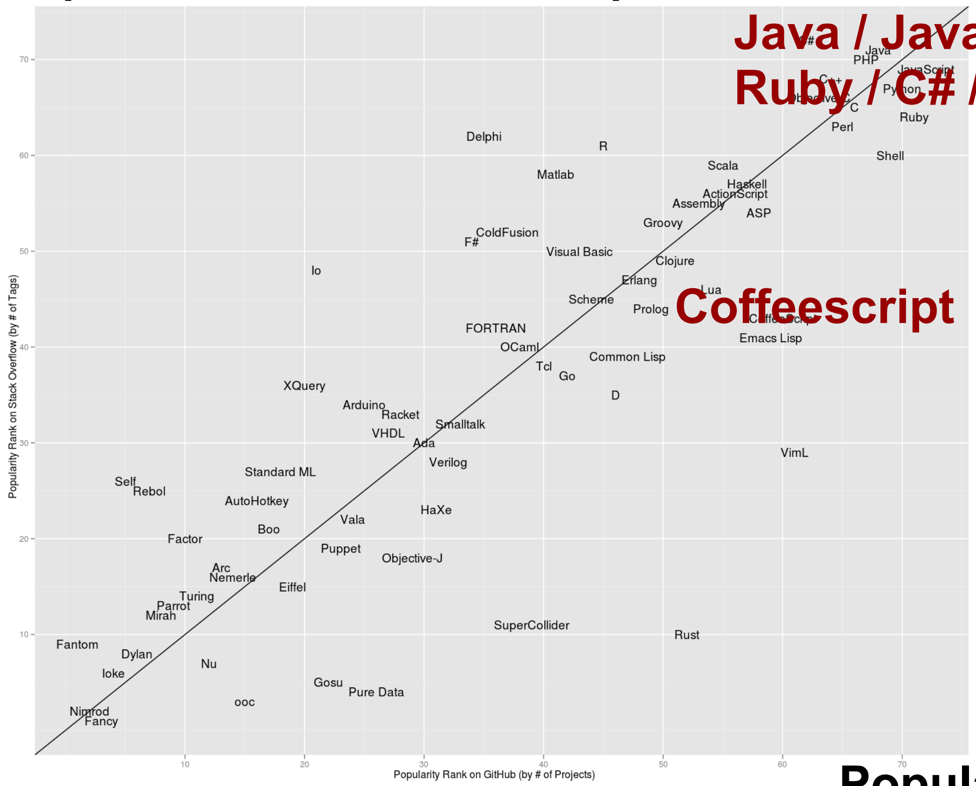
Popularity

- 2009 created (by Jeremy Ashkenas)
- August 2012 - 11th most popular on Github
- Noteable use: Dropbox converted full web client code base
    - dropped 5K LOC (21% reduction)

How popular?

February 2011 Language Popularity (source: RedMonk)

Popularity on Stack Overflow



Java / JavaScript / Ruby / C# / etc

Coffeescript

Popularity on github

From http://www.readwriteweb.com/hack/2012/02/redmonk-programming-language-r.php

Why CoffeeScript?

Unearthing the excellence in JavaScript
JavaScript: The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford

The Case for CoffeeScript

- Protects you from Javascript (no var keyword, no == / type coercion, passes JSLint, wraps blocks in function, etc)

- Expressive / succinct syntactical sugar

- Cool language features (list comprehensions, destructuring assignment, etc) that help with concision

Clean Syntax

```
likeOrUnlike = (req, res) ->
  {itemId, action} = req.body
  userId = Number User.getFromReq(req)?.id

  if not userId or isNaN userId
     return res.json {'error': 'Not logged in'}
magic()
return res.json {success: 'OK'}
```

Compiles into....

Clean Syntax

```
var likeOrUnlike = function(req, res) {
  var action, itemId, userId, _ref, _ref1;
  _ref = req.body, itemId = _ref.itemId, action = _ref.action;
  userId = Number((_ref1 = User.getFromReq(req)) != null ?
_ref1.id : void 0);
  if (!userId || isNaN(userId)) {
    return res.json({
      'error': 'Not logged in'
    });
   magic();
   return res.json({'success': 'OK'})
```

New Feature: List Comprehensions

```
z = (x*x for x in [1,2,3])
```

compiles into:

```
var x, z;
z = (function() {
  var _i, _len, _ref, _results;
  _ref = [1, 2, 3];
  _results = [];
  for (_i = 0, _len = _ref.length; _i < _len; _i++) {
    x = _ref[_i];
    _results.push(x * x);
  }
  return _results;
})();
```

Destructuring Assignment

```
x = {
  name: 'Tony',
  drinks: ['coke', 'sprite'],
  age: 22
}

{name, drinks: [drinkOne, drinkRest...], age} = x

console.log name
console.log drinkOne
console.log drinkRest
console.log age
```

Existential Operator

- x?
  - checks that its not undefined nor null

- a?.b()?.y
  - if a or a.b() is null/undefined, entire expression is false
  - can never have "null pointer exceptions"

So much more

- Classes
- a lot of "shortcuts": e.g., z = {x, y}, no commas for lists separate don newlines, etc.
- string interpolation / block strings
- syntax for binding "this" to funcions
- multi-line regexps
- loops through objects
- etc etc

Trying it out yourself

- http://coffeescript.org/
  - Great examples of features
  - Shows JS -> CS

- Click "Try coffeescript" to see compilation

- run "coffee" for a repl
  - npm install -g coffee

Using with Node

1. npm install -g coffee (and add to package json)
2. create app.js with this content:

```
require('coffee-script');
require('actual-app'); // references actual-app.coffee
```

3. node app.js

- now, *all* files can be written in coffee
- alternatively, can compile with coffee -c x.coffee

Case Against Coffee

- Another layer of abstraction from what's actually going on
  - On the other hand, we don't program in assembly
- In theory, harder to debug since it's compiled
  - Hasn't been an issue for me since I find the conversion to be pretty close
- Relaxed syntax can lead to ambiguous / incorrect code
  - e.g. x y z, w = x(y(z,w)) or x(y(z), w) ?

Links

- Coffeescript homepage
    - http://jashkenas.github.com/coffee-script/
- Dropbox blogpost about switching to CS:
    - https://tech.dropbox.com/?p=361
- Bunch of useful CS links:
    - https://gist.github.com/2764497

Hands-On

git clone git://github.com/jeremyis/coffeescript-hands-on.git

open README.md