JavaScript is for functions

Blatantly stealing from "Hey Underscore, You're Doing It Wrong!" — but I don't like YDIWs

What is JavaScript

* JS is a programming language that runs in browsers
* (Mostly) functional

(Mostly)

* Increased through libs:

UNDERSCORE.JS

What's Underscore?

* A library that adds functional programming stuff to JS
* Collections: array, objects, etc.

What's functional programming?

* Referential transparency:

```
// f(2) → 4
f(2) == 4 // in all cases
```

* Yes to (pure) functions: sqrt, Array
* No to (impure) functions: Date, Math.random

What's functional programming?

* Value-based: everything is an expression (yielding a value)
* Equational (LHS/RHS of function definitions are equivalent values):
  * `var double = function(n) { return 2*n; }`

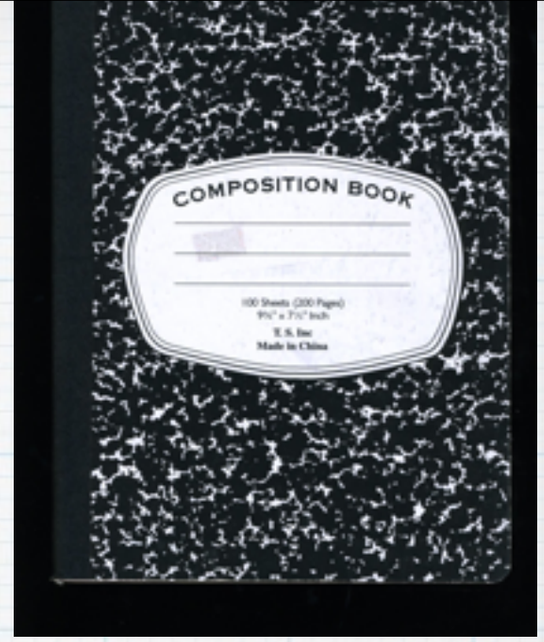
Is Underscore functional?

* Kinda
* It includes the usual "functional friends", **map**, **filter**, **reduce**
* But they *don't* encourage a functional style

Functional Style

* composition
* currying
* use the above for implicit arguments/ specialized functions
* Build abstractions from... wait for it... FUNCTIONS!

Composition




```
// type String → String
function capitalize(s) {
  return s.charAt(0).toUpperCase();
}
```

```
// type String → String
function reverse(s) {
  return s.split(“”).reverse().join("");
}

// type String → String
compose(reverse, capitalize, reverse);
```

Currying

* Allows an N-ary function to be called with <N args.
* ```
  function foo(x, y, z) {…}.autoCurry()
  foo(1, 2, 3) == foo(1)(2)(3)
  // or foo(1,2)(3)
  // or foo(1)(2,3)
  ```
  ```

Currying

```
startsWith = function(letter, word) {
  return letter == word.charAt(0);
}

var pets = [“bird”, “cat”, “dog”];
filter(startsWith(“c”), pets);

// => [“cat”]
```

Functors

* function "g": apply g to a value
* functor "f": apply f to value in a context
* ```
  g(2)    // =>  4
  f([2]) // => [4]
  ```
  ```

```
1  function MyObj(val) {
2    this.val = val;
3  }
4
5  // (a -> b) -> MyObj(a) -> MyObj(b)
6  MyObj.prototype.fmap = function(f) {
7    return new MyObj(f(this.val));
8  }
9
10 // (a -> b) -> f a -> f b
11 function fmap(f, obj) {
12   return obj.fmap(f);
13 }
14
15 // (a -> b) -> [a] -> [b]
16 Array.prototype.fmap = function(f) {
17   return this.map(f);
18 }
19
20 var double = function(n){return n+n};
21
22 fmap(double, new MyObj(1)); // => { val: 2 }
23 fmap(double, [1, 2, 3]);    // => [ 2, 4, 6 ]
~
NORMAL  bar.js  double        unix < utf-8 < javascript  91%  ㏑  21:1
```

Functors

Monads

* NOT scary.
* For composing "lumpy" functions

Lumpy Functions

* Problem: all the words in all the links

```
* function getLinks() {
    return $(‘a').map(function(_,v){
        return v.text;
    });
}
function words(s) {
  return s.split(/\s+/);
}
// compose(words, getLinks)
// => nope :(
```

Waaahhh, I want my compose!

* `getLinks :: [String]`
* `words :: String → [String]`
  * **map** is *almost* it:

```
map :: (String → [String])
    → [String]
    → [[String]]
```

Monads

* They give us back our compose(ure)
* `bind :: (a → m b) → m a → m b`

Monads

* In our case:

```
bind :: (String → [String])
        → [String]
        → [String]
bind(words, getLinks())
```

List monad

1. vim

bash | vim

```
1  // "wrap" a value in the monad
2  Array.prototype.unit = function(x) {
3    return [x];
4  }
5
6  // fmap is just map for arrays
7  Array.prototype.fmap = function(f) {
8    return this.map(f);
9  }
10
11 // join :: m (m a) -> m a, "flatten"
12 Array.prototype.join = function() {
13   return this.reduce(function(xs, x) {
14     return xs.concat(x);
15   }, []);
16 }
17
18 // bind is join(fmap(f, this)) for *any* monad
19 Array.prototype.bind = function(f) {
20   return (this.fmap(f)).join();
21 }
22
23 function words(s){return s.split(/\s+/)}
24 function getLinks() {...} // jQuery stuff
25
26 function bind(f, ma) { return ma.bind(f); }
~
NORMAL  monad.js  getLinks  unix < utf-8 < javascript  96%  25:1
"monad.js" 26L, 575C written
```

Promises

* are a monad for dealing with async programming
* problem with callbacks: sucky type
  * type: aCallback(val, function(){...})
    a → (a → b) → null?

Promises

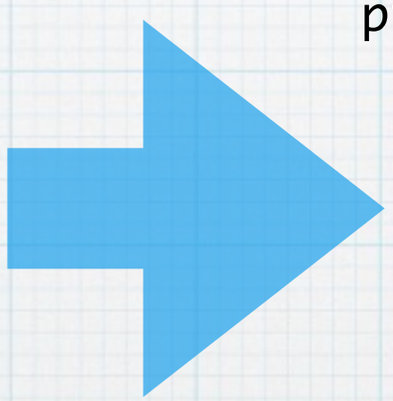
* Type: a → Promise b

* look familiar?
  bind :: (a → m b) → m a → m b

  (m = "Promise")

Promise Example

```
// readFile :: String -> Promise String
var readFile = function(path) {
  var promise = new Promise();
  fs.readFile(path, function(err, content) {
    promise.succeed(content);
  });
  return promise;
};

// getUrl :: String -> Promise URI
var getUrl = function(json) {
  var uri = url.parse(JSON.parse(json).url);
  return new Promise(uri);
};

// httpGet :: URI -> Promise Response
var httpGet = function(uri) {
  var client  = http.createClient(80, uri.hostname),
      request = client.request('GET', uri.pathname, {'Host': uri.hostname}),
      promise = new Promise();

  request.addListener('response', function(response) {
    promise.succeed(response);
  });
  request.end();
  return promise;
};

// responseBody :: Response -> Promise String
var responseBody = function(response) {
  var promise = new Promise(),
      body    = '';

  response.addListener('data', function(c) { body += c });
  response.addListener('end', function() {
    promise.succeed(body);
  });
  return promise;
};

// print :: String -> Promise null
var print = function(string) {
  return new Promise(sys.puts(string));
};
```



```
pipe(unit(__dirname + '/urls.json'),
     [ readFile,
       getUrl,
       httpGet,
       responseBody,
       print          ]);
```

Example from: jcoglan

Yay, Promises! Yay?

* Well...
* jQuery promises are surely broken :(
* Not sure what's best (Q.js maybe?)
  * I haven't studied this extensively
  * I'll get back to you :)

But...

* Some people have difficulty seeing benefits
* High-level, algebraic JS is GREAT!
* Functions for free (— or at least low prices)
* Future isn't widely distributed yet

Mystery BONUS Slide(s)

* Biggest problems in Computer Science
  * Cache invalidation
  * Naming things

Mystery BONUS Slide(s)

* Pure functions
  * No notion of time or state
  * Cache invalidation: SOLVED!

Mystery BONUS Slide(s)

* Coding with composition of functions
  * No variable names!
    * ummm, still function names :(
  * Naming things: ½ SOLVED!

Mystery BONUS Slide(s)

* Functional programming:
* 1½ *hardest* problems in CS
  * SOLVED!

You dig?

* Want to jam with the console cowboys in cyberspace?
* Well then...

Fantasy Land

* Defines a bunch of algebraic structs in JS!
* great argument: https://github.com/promises-aplus/promises-spec/issues/94

README.md

Fantasy Land Specification

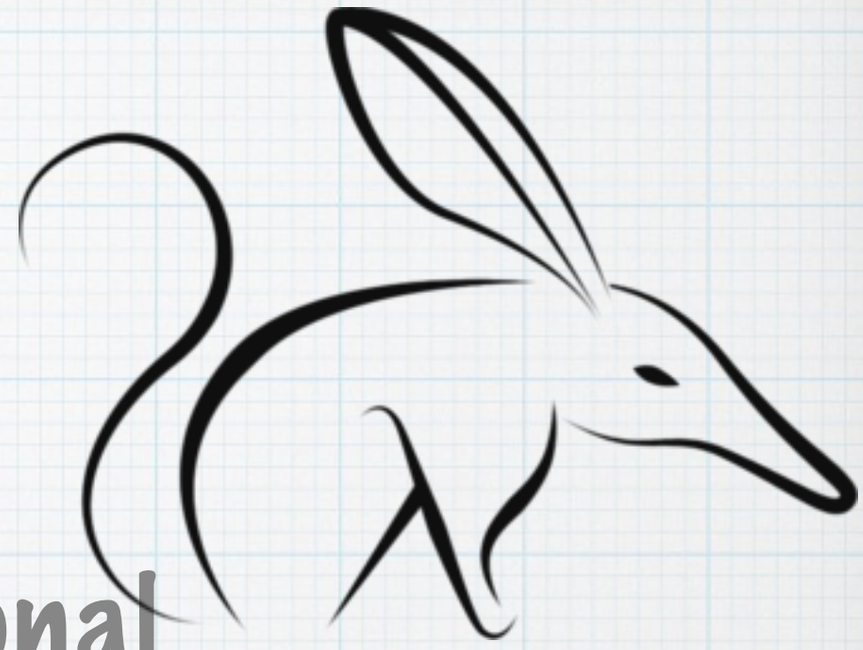
(aka "Algebraic JavaScript Specification")

Functional Javascript

* http://osteele.com/sources/javascript/functional/
* Really succinct JS functions:

```
map('x*x', [1,2,3,4])
```

bilby.js



* I quote: "...a serious functional programming library. Serious, meaning it applies category theory to enable highly abstract and generalised code. Functional, meaning that it enables referentially transparent programs."

* http://bilby.brianmckenna.org

Underscore...YDIW

* Don't like YDIWs... but cool info
* http://www.youtube.com/watch?v=m3svKOdZijA

Thanks!