CODE 301

Intermediate Software Development

FUNCTIONAL PROGRAMMING

SIMPLE != EASY

- Rich Hickey

"SOMETIMES, THE ELEGANT IMPLEMENTATION IS JUST A FUNCTION. NOT A METHOD. NOT A CLASS. NOT A FRAMEWORK. JUST A FUNCTION."

- John Carmack

FUNCTIONAL PROGRAMMING

- Why?
  - Long tradition going back to Lisp (vs Fortran)
  - Has been primarily in academia but strongly resurgent in industry
  - For many hiring managers, a signal that you know what you're doing
  - Cleaner code - easier to reason about
  - Scalable and Performant on multi-core systems, large volumes of data

HOW FUNCTIONAL PROGRAMMING RELATES TO JS

- JavaScript was almost Scheme, a Functional language based on LISP
- In 1994 Brendan Eich wanted a functional language for the browser but under competitive pressure from Java introduced OO features into the language
- But at its core, JS is still a Functional Language. Why?

VARIABLES ARE SCOPED TO FUNCTIONS IN JS

- The only place to "attach" a variable in JavaScript is to a function, with the **`var`** keyword

```
var window_scope_var = 'you can find me at window.window_scope_var'

function doSomething() {
  var function_scope = 'you can only use me in the scope of
function'
  console.log(window_scope_var);

  function innerFunction() {
    var inner_scope = 'I am only in this function!'
    console.log(function_scope);
    console.log(inner_scope);
  }
}
```

WHAT IS FUNCTIONAL PROGRAMMING

- What is it? No one “standard” for functional but includes:
  - Immutability
  - Declarative vs. Imperative code
  - Stateless (pure) functions
  - First-class Functions and Currying

FUNCTIONAL PROGRAMMING

- Functional features built in to JavaScript (ECMA 5 standard)
  - Array
    - .forEach
    - .some and .every
    - .concat
    - .filter
    - .map
    - .reduce

MUTABILITY AND IMMUTABILITY

- A fancy way of saying "changeable"
- For example, Array Methods:

- Don't Mutate the data
  - forEach
  - Slice
  - Map
  - Reduce
  - Filter

- Mutate the data
  - Sort
  - Reverse
  - Splice

IMMUTABILITY

- Why?
  - Limiting the amount of things that change gives focus
  - Take away opportunities for things to be unintentionally modified
- Cons
  - Harder, (but simpler). Memory usage (maybe)
- “Shared mutable state is the root of all evil.” - Pete hunt
- There are libraries for immutability in JS, but not required
  - ImmutableJS, Mori, Deep-freeze
- Object.freeze()

THE TAKEAWAY?

Look for opportunities in your code to remove changing a variable's value - mutable state.

DECLARATIVE VS IMPERATIVE

- Describe **WHAT** you want

vs

- **HOW**: The steps to get it done

IMPERATIVE EXAMPLE

$$s = \sum_{x=1}^{N} x^2 = 1^2 + 2^2 + 3^3 + \ldots + N^2$$

```
function sumOfSquares(nums) {
  var i, sum = 0, squares = [];
  for (i = 0; i < nums.length; i++) {
    squares.push(nums[i]*nums[i]);
  }

  for (i = 0; i < squares.length; i++) {
    sum += squares[i];
  }

  return sum;
}

console.log(sumOfSquares([1, 2, 3, 4, 5]));
```

DECLARATIVE EXAMPLE

$$s = \sum_{x=1}^{N} x^2 = 1^2 + 2^2 + 3^3 + \ldots + N^2$$

```
function sumOfSquares2(nums) {
  return nums
    .map(function(num) { return num * num; })
    .reduce(function(start, num) { return start + num; }, 0)
    ;
}

console.log(sumOfSquares2([1, 2, 3, 4, 5]));
```

PURE (STATELESS) FUNCTIONS

```
// pure (stateless)

function square(x) {
  return x * x;
}

function squareAll(items) {
  return items.map(square);
}
```

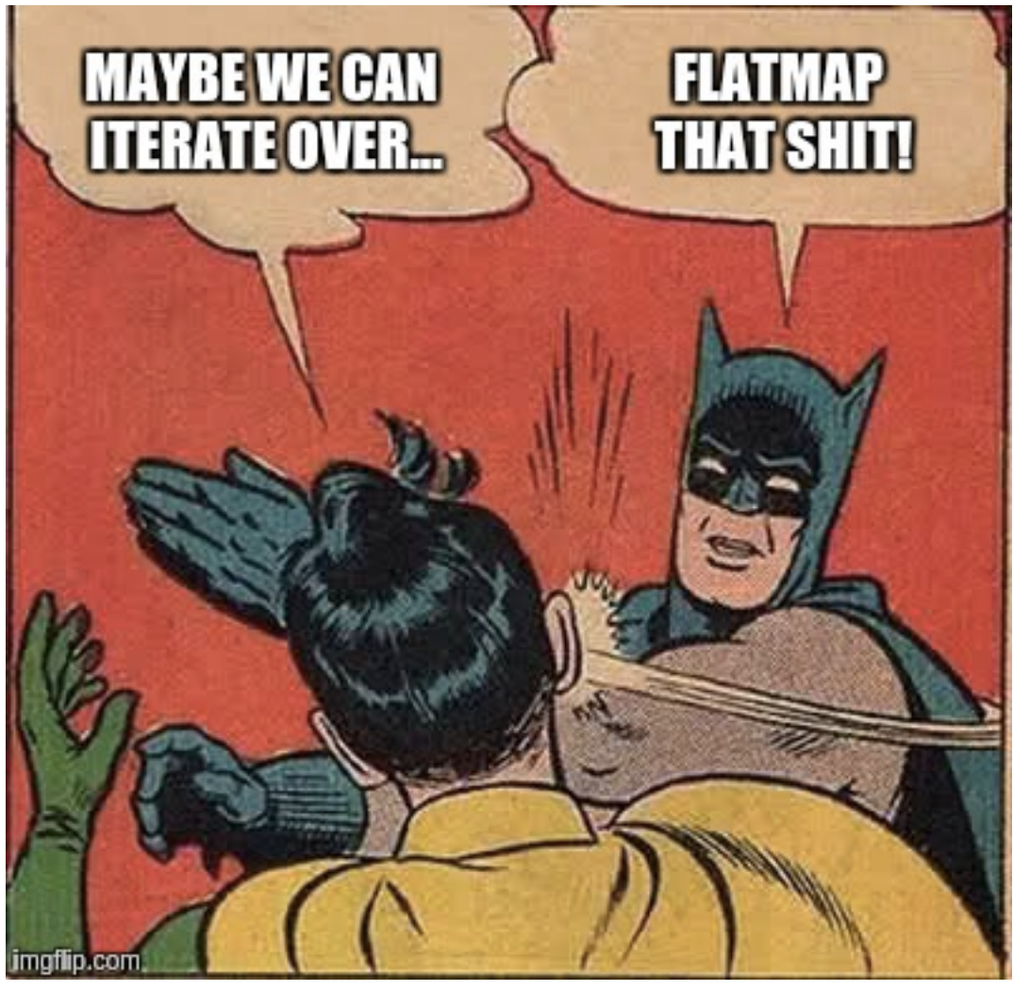
```
//  impure (stateful)

function square(x) {
  updateXinDatabase(x);
  return x * x;
}

function squareAll(items) {
  var i;
  for (i = 0; i < items.length; i++) {
    items[i] = square( items[i] );
  }
}
```

FUNCTIONAL PROGRAMMING

- There's much more to discover!
  - https://lodash.com
  - https://drboolean.gitbooks.io/mostly-adequate-guide/
  - http://reactivex.io/learnrx/
  - http://www.infoq.com/presentations/Simple-Made-Easy



- Predicates, Optionals, Functors, oh my!