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

- **The Why:**
  - Functional programming concepts have been primarily in academia but strongly resurgent in the industry.
  - Effects + Logic = Side Effects.
  - Cleaner code
    - Easier to read, modify, and debug.
  - Scalable on multi-core systems, large volumes of data.

WHAT IS FUNCTIONAL PROGRAMMING

- **The What:**
  - While there isn’t a defined list of what makes something part of a functional programming paradigm, the following items can be included in this concept:
    - Immutability (Strings, Numbers)
    - Declarative vs. Imperative code
    - Stateless (pure) functions
    - First-class Functions

FUNCTIONAL PROGRAMMING

- Functional features built in to JavaScript
  - Array Methods
    - .forEach *(applies function once per element)*
    - .some and .every *(returns a boolean)*
    - **.concat** *(returns new array - think push(), but without side-effects!)*
    - **.filter** *(returns new array of *values **based** on boolean results)*
    - **.map** *(returns new array of values based on the function applied)*
    - **.reduce** *(return new value based on the accumulator set)*

MUTABILITY AND IMMUTABILITY

- A fancy way of saying "changeable"
- For example, Array Methods:
- Don't Mutate the data
  - forEach
  - Slice
  - Map
  - Filter
  - Reduce
- Mutate the data
  - Sort
  - Reverse
  - Splice

IMMUTABILITY

- **A Few More Reasons:**
  - Limits the amount of things that change (reduces risk).
  - Takes away opportunities for things to be unintentionally modified.
- Trade-offs
  - Harder, (but simpler). Memory usage (maybe)
- There are libraries for immutability in JS, but not required
  - ImmutableJS, Mori, Deep-freeze

DECLARATIVE VS IMPERATIVE

- Describe **WHAT** you want (declarative)

vs

- **HOW**: The steps to get it done (imperative)

- Declarative: *“I want a cookie!”*

- Imperative: *“Head to Macrina Bakery ….” etc.*

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 sumOfSquaresDeclarative(nums) {
  return nums
    .map(function(num) { return num * num; })
    .reduce(function(prev, cur) { return prev + cur; }, 0)
    ;
}

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

PUSH: UNDER THE HOOD????

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

```
function myPush(array) {
  for (var i = 1; i < arguments.length; i++) {
    array[array.length] = arguments[i];
  }
  return array.length;
}
```

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] );
  }
}
```

FIRST CLASS FUNCTIONS

- Also called higher-order functions or λ
- In JS, all functions are objects
- You’ve already been using these in callbacks, etc.
- Enable Abstraction and Composability

```
// pure (stateless)

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

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

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

RECAP

RECAP

“Functional programming will make your programs more understandable, maintainable, and reliable.”