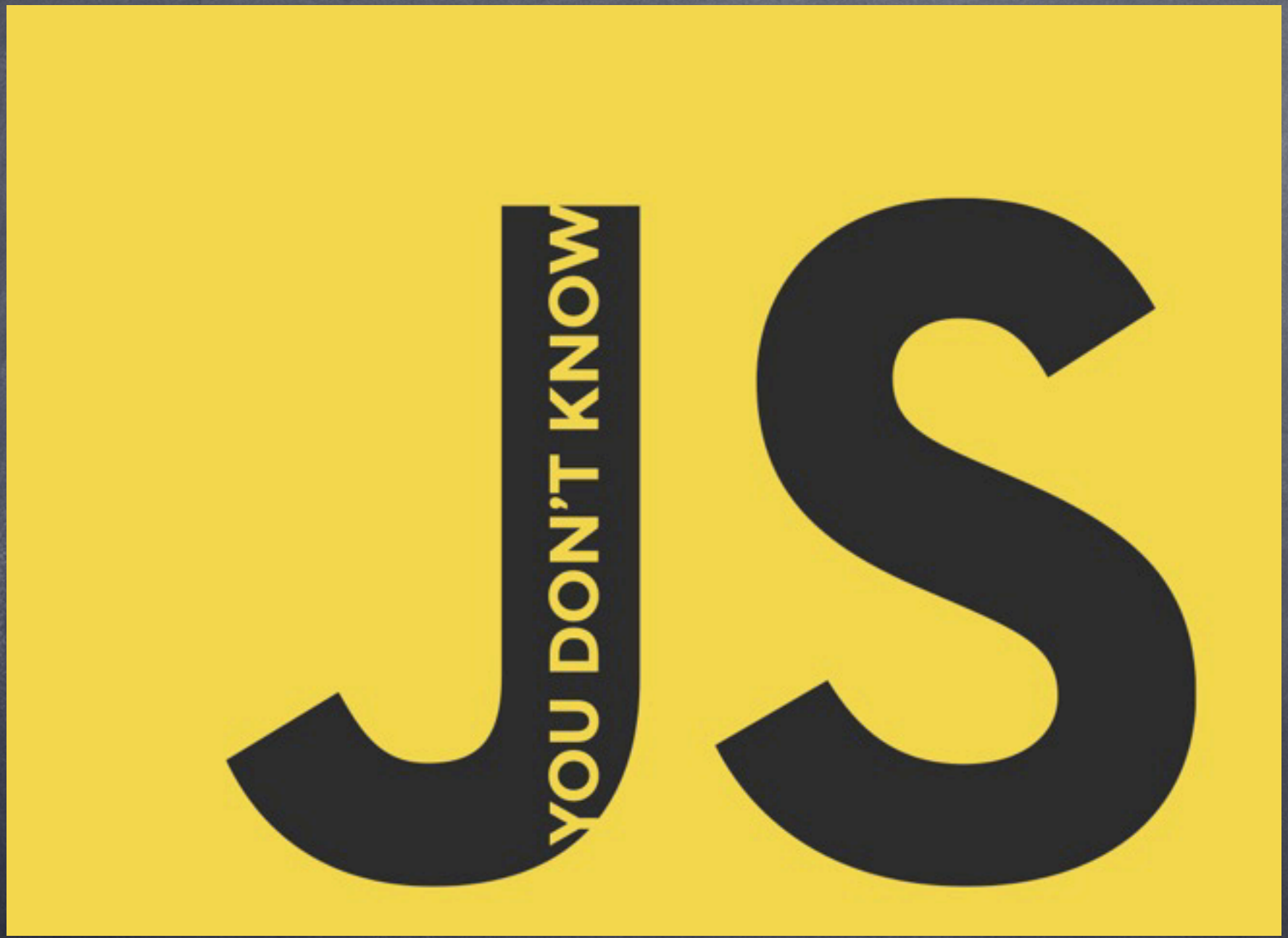
Kyle Simpson
@getify
http://getify.me

- LABjs
- grips
- asynquence


MAKER
SQUARE



http://YouDontKnowJS.com

Agenda

Functional-Light Programming

- Pure Functions
- Composition
- Immutability
- Closure
- Recursion
- List Transformation (map)
- List Exclusion (filter)
- List Composition (reduce)
- List Iteration (forEach)

Side Effects

```
 1 function foo(x) {
 2     y = x * 2;
 3     z = x * 3;
 4 }
 5
 6 var y, z;
 7
 8 foo(5);
 9
10 y;   // 10
11 z;   // 15
```

Functional-Light Programming: side effects

```
1  function foo(x) {
2      y = y * x;
3      z = y * x;
4  }
5
6  var y = 2, z = 3;
7
8  foo(5);
9
10 y;   // 10
11 z;   // 15
12
13 foo(5);
14
15 y;   // 50
16 z;   // 75
```

Functional-Light Programming: side effects

Pure: No Side Effects

```
1  function bar(x,y,z) {
2      foo(x);
3      return [y,z];
4
5      function foo(x) {
6          y = y * x;
7          z = y * x;
8      }
9  }
10
11 bar(5,2,3);      // [10,15]
12
13 bar(5,10,15);    // [50,75]
```

Functional-Light Programming: pure functions

(exercise #1: 5min)

Composition

```
 1 function sum(x,y) {
 2     return x + y;
 3 }
 4
 5 function mult(x,y) {
 6     return x * y;
 7 }
 8
 9 // 5 + (3 * 4)
10 var z = mult(3,4);
11 z = sum(z,5);
12
13 z;          // 17
```

Functional-Light Programming: composition

```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  // 5 + (3 * 4)
10 sum( mult(3,4), 5 );    // 17
```

Functional-Light Programming: composition

```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  function multAndSum(x,y,z) {
10     return sum( mult(x,y), z );
11 }
12
13 // 5 + (3 * 4)
14 multAndSum(3,4,5);   // 17
```

Functional-Light Programming: composition

```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  function compose2(fn1,fn2) {
10     return function comp(){
11         var args = [].slice.call(arguments);
12         return fn2(
13             fn1(args.shift(),args.shift()),
14             args.shift()
15         );
16     }
17 }
18
19 var multAndSum = compose2(mult,sum);
20
21 multAndSum(3,4,5);      // 17
```

Functional-Light Programming: composition

Immutability

```
 1 var x = 2;
 2 x++;                    // allowed
 3
 4 const y = 3;
 5 y++;                    // not allowed!
 6
 7 const z = [4,5,6];
 8 z = 10;                 // not allowed!
 9 z[0] = 10;              // allowed!
10
11 const w = Object.freeze([4,5,6]);
12 w = 10;                 // not allowed!
13 w[0] = 10;              // not allowed!
```

Functional-Light Programming: immutability

```
1  function doubleThemMutable(list) {
2      for (var i=0; i<list.length; i++) {
3          list[i] = list[i] * 2;
4      }
5  }
6
7  var arr = [3,4,5];
8  doubleThemMutable(arr);
9
10 arr;     // [6,8,10]
```

Functional-Light Programming: immutability

```
1  function doubleThemImmutable(list) {
2      var newList = [];
3      for (var i=0; i<list.length; i++) {
4          newList[i] = list[i] * 2;
5      }
6      return newList;
7  }
8
9  var arr = [3,4,5];
10 var arr2 = doubleThemImmutable(arr);
11
12 arr;     // [3,4,5]
13 arr2;    // [6,8,10]
```

Functional-Light Programming: immutability

Closure

Closure is when a function "remembers" the variables around it even when that function is executed elsewhere.

```
1  function foo() {
2      var count = 0;
3
4      return function(){
5          return count++;
6      };
7  }
8
9  var x = foo();
10
11 x();     // 0
12 x();     // 1
13 x();     // 2
```

Functional-Light Programming: closure

```
1  function sumX(x) {
2      return function(y) {
3          return x + y;
4      };
5  }
6
7  var add10 = sumX(10);
8
9  add10(3);          // 13
10 add10(14);         // 24
```

Functional-Light Programming: closure

(exercise #2: 5min)

Recursion

```
1  function sumIter() {
2      var sum = 0;
3      for (var i=0; i<arguments.length; i++) {
4          sum = sum + arguments[i];
5      }
6      return sum;
7  }
8
9  sumIter(3,4,5);      // 12
```

Functional-Light Programming: recursion

```
1  function sumRecur() {
2      var args = [].slice.call(arguments);
3      if (args.length <= 2) {
4          return args[0] + args[1];
5      }
6      return (
7          args[0] +
8          sumRecur.apply(null,args.slice(1))
9      );
10 }
11
12 sumRecur(3,4,5);          // 12
```

Functional-Light Programming: recursion

ES6 ftw!

```
1  function sumRecur(...args) {
2      if (args.length <= 2) {
3          return args[0] + args[1];
4      }
5      return (
6          args[0] +
7          sumRecur(...args.slice(1))
8      );
9  }
10
11 sumRecur(3,4,5);          // 12
```

Functional-Light Programming: recursion

(exercise #3: 5min)

List Transformation (map)

```
1  function doubleIt(v) { return v * 2; }
2
3  function transform(arr,fn) {
4      var list = [];
5      for (var i=0; i<arr.length; i++) {
6          list[i] = fn(arr[i]);
7      }
8      return list;
9  }
10
11
12 transform([1,2,3,4,5],doubleIt);
13 // [2,4,6,8,10]
```

Functional-Light Programming: transformation

```
1 function doubleIt(v) { return v * 2; }
2
3 [1,2,3,4,5].map(doubleIt);
4 // [2,4,6,8,10]
```

Functional-Light Programming: transformation

List Exclusion (filter)

```
1  function isOdd(v) { return v % 2 == 1; }
2
3  function exclude(arr,fn) {
4      var list = [];
5      for (var i=0; i<arr.length; i++) {
6          if (fn(arr[i])) {
7              list.push(arr[i]);
8          }
9      }
10     return list;
11 }
12
13
14 exclude([1,2,3,4,5],isOdd);
15 // [1,3,5]
```

Functional-Light Programming: exclusion

```
1 function isOdd(v) { return v % 2 == 1; }
2
3 [1,2,3,4,5].filter(isOdd);
4 // [1,3,5]
```

Functional-Light Programming: exclusion

List Composition (reduce)

```
1  function mult(x,y) { return x * y; }
2
3  function compose(arr,fn,initial) {
4      var total = initial;
5      for (var i=0; i<arr.length; i++) {
6          total = fn(total,arr[i]);
7      }
8      return total;
9  }
10
11
12 compose([1,2,3,4,5],mult,1);
13 // 120
```

Functional-Light Programming: composition

```
1 function mult(x,y) { return x * y; }
2
3 [1,2,3,4,5].reduce(mult,1);
4 // 120
```

Functional-Light Programming: composition

List Iteration (!!)
(forEach)

```
1  function logValue(v) { console.log(v); }
2
3  function iterate(arr,fn) {
4      for (var i=0; i<arr.length; i++) {
5          fn(arr[i]);
6      }
7  }
8
9
10 iterate([1,2,3,4,5],logValue);
11 // 1 2 3 4 5
```

Functional-Light Programming: iteration

```
1 function logValue(v) { console.log(v); }
2
3 [1,2,3,4,5].forEach(logValue);
4 // 1 2 3 4 5
```

Functional-Light Programming: iteration

(exercise #4: 30min)

Kyle Simpson
@getify
http://getify.me

Thanks!

Questions?