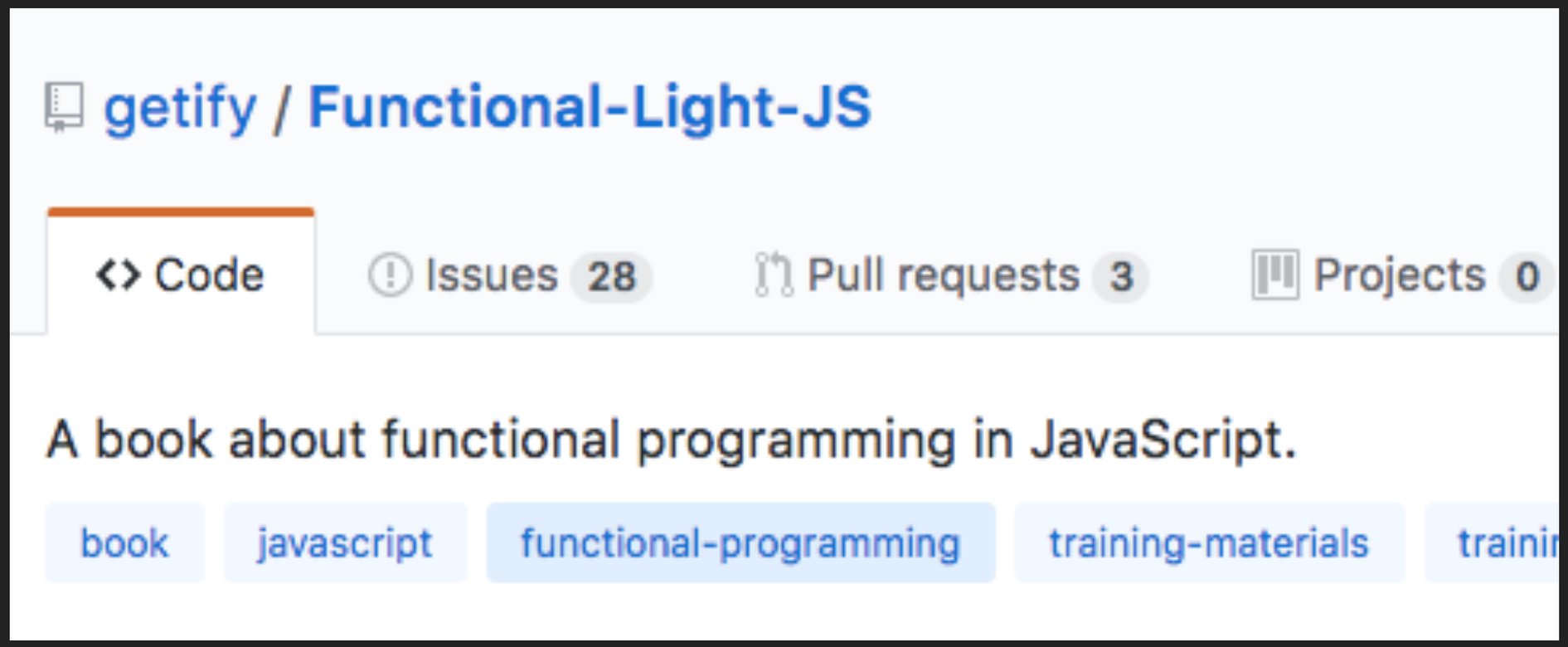
KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS



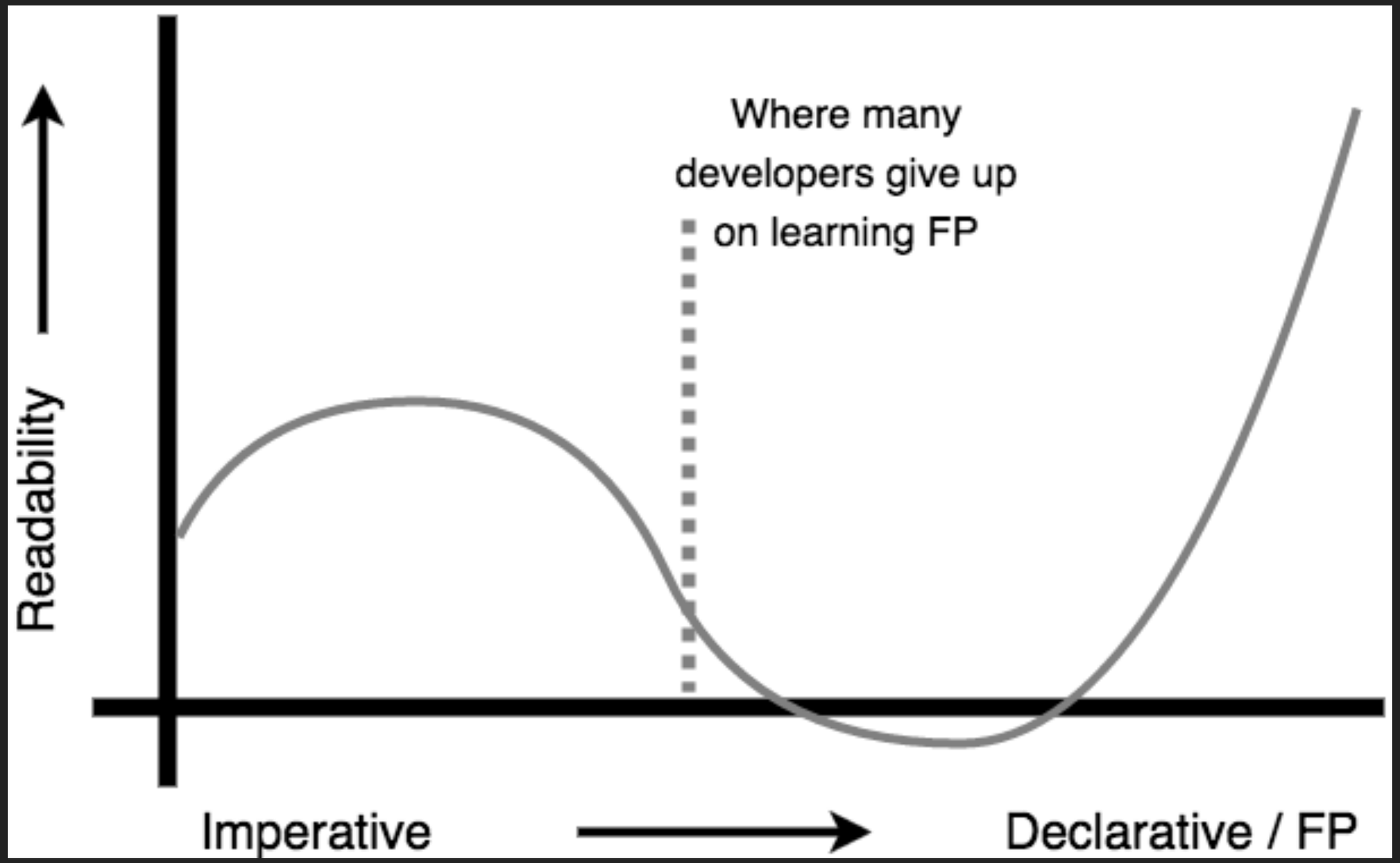
github.com/getify/Functional-Light-JS

WHY FP?

IMPERATIVE

VS

DECLARATIVE


Readability
Where many
developers give up
on learning FP
Imperative
Declarative / FP

PROVABLE

LESS TO READ

Course Overview

- Functions
- Closure
- Composition
- Immutability
- Recursion
- Lists / Data Structures
- Async
- FP Libraries

...but before we begin...

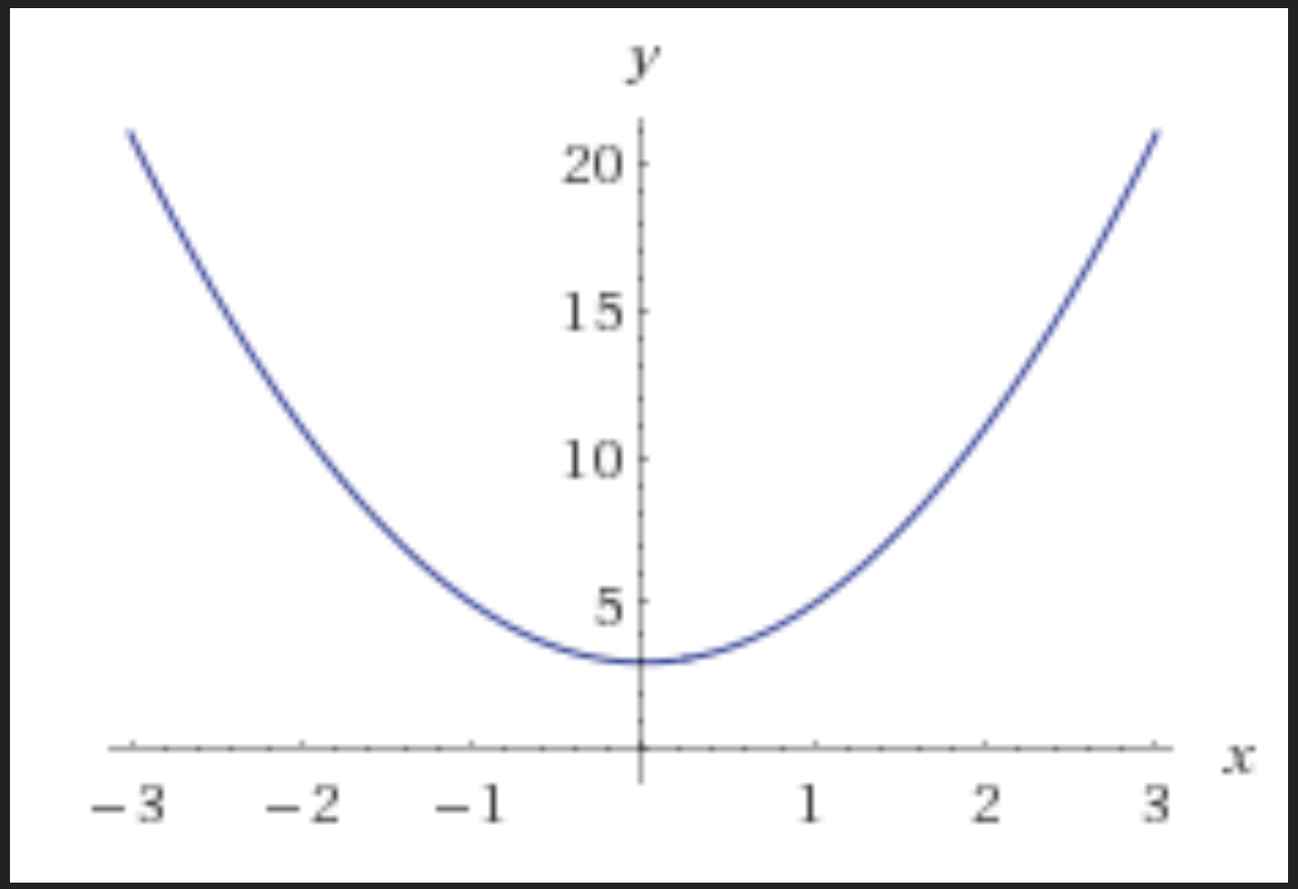
FUNCTIONS

Procedures

```
function addNumbers(x = 0,y = 0,z = 0,w = 0) {
    var total = x + y + z + w;
    console.log(total);
}

function extraNumbers(x = 2,...args) {
    return addNumbers(x,40,...args);
}


extraNumbers();          // 42

extraNumbers(3,8,11);    // 62
```

Functions

```
1 function tuple(x,y) {
2     return [x + 1, y - 1];
3 }
4
5 var [a,b] = tuple(...[5,10]);
6
7 a;                // 6
8 b;                // 9
```



$$f(x) = 2x^2 + 3$$

```
function f(x) {
    return 2 * Math.pow(x,2) + 3;
}
```

Function: the semantic relationship between input and computed output

```
1 function shippingRate(size,weight,speed) {
2     return ((size + 1) * weight) + speed;
3 }
```

~~SIDE EFFECTS~~

```
function shippingRate() {
    rate = ((size + 1) * weight) + speed;
}

var rate;
var size = 12;
var weight = 4;
var speed = 5;
shippingRate();
rate;                    // 57

size = 8;
speed = 6;
shippingRate();
rate;                    // 42
```

```
1 function shippingRate(size,weight,speed) {
2     return ((size + 1) * weight) + speed;
3 }
4
5 var rate;
6
7 rate = shippingRate(12,4,5);     // 57
8
9 rate = shippingRate(8,4,6);      // 42
```

Avoid side effects with function calls... if possible

Side Effects:

- I / O (console, files, etc)
- Database Storage
- Network Calls
- DOM
- Timestamps
- Random Numbers
- CPU Heat
- CPU Time Delay
- etc

No such thing as
"no side effects"

Avoid them where possible,
make them obvious otherwise

PURE FUNCTIONS

```
1 // pure
2 function addTwo(x,y) {
3     return x + y;
4 }
5
6 // impure
7 function addAnother(x,y) {
8     return addTwo(x,y) + z;
9 }
```

```
1 var z = 1;
2
3 function addTwo(x,y) {
4     return x + y;
5 }
6
7 function addAnother(x,y) {
8     return addTwo(x,y) + z;
9 }
10
11 addAnother(20,21);  // 42
```

```
1  var z = 1;
2
3  function addTwo(x,y) {
4      return x + y;
5  }
6
7  function addAnother(x,y) {
8      return addTwo(x,y) + z;
9  }
10
11 addAnother(20,21);  // 42
```

```
1 function addAnother(z) {
2     return function addTwo(x,y) {
3         return x + y + z;
4     };
5 }
6
7 addAnother(1)(20,21);   // 42
```

```
1 function getId(obj) {
2         return obj.id;
3 }
4
5 getId({
6         get id() {
7                 return Math.random();
8         }
9 });
```

Function Purity: Level of Confidence

Function Purity: Calls, Not Definitions

EXTRACTING IMPURITY

```
1  function addComment(userID,comment) {
2      var record = {
3          id: uniqueID(),
4          userID,
5          text: comment
6      };
7      var elem = buildCommentElement(record);
8      commentsList.appendChild(elem);
9  }
10
11 addComment(42,"This is my first comment!");
```

```
1  function newComment(userID,commentID,comment) {
2      var record = {
3          id: commentID,
4          userID,
5          text: comment
6      };
7      return buildCommentElement(record);
8  }
9
10 var commentID = uniqueID();
11 var elem = newComment(
12     42,
13     commentID,
14     "This is my first comment!"
15 );
16 commentsList.appendChild(elem);
```

CONTAINING IMPURITY

```
1  var SomeAPI = {
2      threshold: 13,
3      isBelowThreshold(x) {
4          return x <= SomeAPI.threshold;
5      }
6  };
7  var numbers = [];
8
9  function insertSortedDesc(v) {
10     SomeAPI.threshold = v;
11     var idx = numbers.findIndex(SomeAPI.isBelowThreshold);
12     if (idx == -1) {
13         idx = numbers.length;
14     }
15     numbers.splice(idx,0,v);
16 }
17
18 insertSortedDesc(3);
19 insertSortedDesc(5);
20 insertSortedDesc(1);
21 insertSortedDesc(4);
22 insertSortedDesc(2);
23 numbers;                    // [ 5, 4, 3, 2, 1 ]
```

```javascript
var SomeAPI = {
    threshold: 13,
    isBelowThreshold(x) {
        return x <= SomeAPI.threshold;
    }
};
var numbers = [];

function getSortedNums(nums,v) {
    var numbers = nums.slice();
    insertSortedDesc(v);
    return numbers;

    function insertSortedDesc(v) {
        SomeAPI.threshold = v;
        var idx = numbers.findIndex(SomeAPI.isBelowThreshold);
        if (idx == -1) {
            idx = numbers.length;
        }
        numbers.splice(idx,0,v);
    }
}

numbers = getSortedNums(numbers,3);
numbers = getSortedNums(numbers,5);
numbers = getSortedNums(numbers,1);
numbers = getSortedNums(numbers,4);
numbers = getSortedNums(numbers,2);
numbers;                                    // [ 5, 4, 3, 2, 1 ]
```

```
1  var SomeAPI = {
2      threshold: 13,
3      isBelowThreshold(x) {
4          return x <= SomeAPI.threshold;
5      }
6  };
7  var numbers = [];
8
9  function insertSortedDesc(v) {
10     SomeAPI.threshold = v;
11     var idx = numbers.findIndex(SomeAPI.isBelowThreshold);
12     if (idx == -1) {
13         idx = numbers.length;
14     }
15     numbers.splice(idx,0,v);
16 }
17
18 function getSortedNums(nums,v) {
19     var [origNumbers,origThreshold] = [numbers,SomeAPI.threshold];
20     numbers = nums.slice();
21     insertSortedDesc(v);
22     nums = numbers;
23     [numbers,SomeAPI.threshold] = [origNumbers,origThreshold];
24     return nums;
25 }
26
27 numbers = getSortedNums(numbers,3);
28 numbers = getSortedNums(numbers,5);
29 numbers = getSortedNums(numbers,1);
30 numbers = getSortedNums(numbers,4);
31 numbers = getSortedNums(numbers,2);
32 numbers;                                  // [ 5, 4, 3, 2, 1 ]
```

1. 19
2. 20
3. 21
4. 22
5. 23
6. 24

ARGUMENTS

```
// unary
function increment(x) {
    return sum(x,1);
}

// binary
function sum(x,y) {
    return x + y;
}
```

```
1  function unary(fn) {
2      return function one(arg){
3          return fn(arg);
4      };
5  }
6
7  function binary(fn) {
8      return function two(arg1,arg2){
9          return fn(arg1,arg2);
10     };
11 }
12
13 function f(...args) {
14     return args;
15 }
16
17 var g = unary(f);
18 var h = binary(f);
19
20 g(1,2,3,4);      // [1]
21 h(1,2,3,4);      // [1,2]
```

```
1  function flip(fn) {
2      return function flipped(arg1,arg2,...args){
3          return fn(arg2,arg1,...args);
4      };
5  }
6
7  function f(...args) {
8      return args;
9  }
10
11 var g = flip(f);
12
13 g(1,2,3,4);     // [2,1,3,4]
```

```javascript
function reverseArgs(fn) {
    return function reversed(...args){
        return fn(...args.reverse());
    };
}

function f(...args) {
    return args;
}

var g = reverseArgs(f);

g(1,2,3,4);     // [4,3,2,1]
```

```
1  function spreadArgs(fn) {
2      return function spread(args){
3          return fn(...args);
4      };
5  }
6
7  function f(x,y,z,w) {
8      return x + y + z + w;
9  }
10
11 var g = spreadArgs(f);
12
13 g([1,2,3,4]);        // 10
```

unspread(..)?

POINT-FREE

```
1 getPerson(function onPerson(person){
2     return renderPerson(person);
3 });
4
5 getPerson(renderPerson);
```

Equational Reasoning

Shape

```
1 function isOdd(v) {
2     return v % 2 == 1;
3 }
4
5 function isEven(v) {
6     return !isOdd(v);
7 }
8
9 isEven(4);        // true
```

```
1  function not(fn) {
2      return function negated(...args){
3          return !fn(...args);
4      };
5  }
6
7  function isOdd(v) {
8      return v % 2 == 1;
9  }
10
11 var isEven = not(isOdd);
12
13 isEven(4);        // true
```

Advanced Point-Free

```
1  function mod(y) {
2      return function forX(x){
3          return x % y;
4      };
5  }
6  function eq(y) {
7      return function forX(x){
8          return x === y;
9      };
10 }
11
12 var mod2 = mod(2);
13 var eq1 = eq(1);
14
15 function isOdd(x) {
16     return eq1( mod2( x ) );
17 }
```

```
1  var mod2 = mod(2);
2  var eq1 = eq(1);
3
4  function isOdd(x) {
5      return eq1( mod2( x ) );
6  }
7
8  function compose(fn2,fn1) {
9      return function composed(v){
10         return fn2( fn1( v ) );
11     };
12 }
13
14 var isOdd = compose(eq1,mod2);
15
16 var isOdd = compose(eq(1),mod(2));
```

CLOSURE

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

```
1  function makeCounter() {
2      var counter = 0;
3      return function increment(){
4          return ++counter;
5      };
6  }
7
8  var c = makeCounter();
9
10 c();     // 1
11 c();     // 2
12 c();     // 3
```

```
1 function unary(fn) {
2     return function one(arg){
3         return fn(arg);
4     };
5 }
```

```
1 function addAnother(z) {
2     return function addTwo(x,y) {
3         return x + y + z;
4     };
5 }
```

LAZY VS EAGER

```
1  function repeater(count) {
2      return function allTheAs(){
3          return "".padStart(count,"A");
4      };
5  }
6
7  var A = repeater(10);
8
9  A();     // "AAAAAAAAAA"
10 A();     // "AAAAAAAAAA"
```

```
1  function repeater(count) {
2      var str = "".padStart(count,"A");
3      return function allTheAs(){
4          return str;
5      };
6  }
7
8  var A = repeater(10);
9
10 A();     // "AAAAAAAAAA"
11 A();     // "AAAAAAAAAA"
```

```
1  function repeater(count) {
2      var str;
3      return function allTheAs(){
4          if (str == undefined) {
5              str = "".padStart(count,"A");
6          }
7          return str;
8      };
9  }
10
11 var A = repeater(10);
12
13 A();     // "AAAAAAAAAA"
14 A();     // "AAAAAAAAAA"
```

Memoization

```
1  function repeater(count) {
2      return memoize(function allTheAs(){
3          return "".padStart(count,"A");
4      });
5  }
6
7  var A = repeater(10);
8
9  A();     // "AAAAAAAAAA"
10 A();     // "AAAAAAAAAA"
```

GENERALIZED TO SPECIALIZED

```
1  function ajax(url,data,cb) { /*..*/ }
2
3  ajax(CUSTOMER_API,{id:42},renderCustomer);
4
5  function getCustomer(data,cb) {
6      return ajax(CUSTOMER_API,data,cb);
7  }
8
9  getCustomer({id:42},renderCustomer);
10
11 function getCurrentUser(cb) {
12     // return ajax(CUSTOMER_API,{id:42},cb);
13     return getCustomer({id:42},cb);
14 }
15
16 getCurrentUser(renderCustomer);
```

Function Parameter Order:

General -> Specific

PARTIAL APPLICATION

```
1 function ajax(url,data,cb) { /*..*/ }
2
3 var getCustomer = partial(ajax,CUSTOMER_API);
4 // var getCurrentUser = partial(ajax,CUSTOMER_API,{id:42});
5 var getCurrentUser = partial(getCustomer,{id:42});
6
7 getCustomer({id:42},renderCustomer);
8
9 getCurrentUser(renderCustomer);
```

CURRYING

```
1  function ajax(url) {
2      return getData(data){
3          return getCB(cb){ /*..*/ };
4      }
5  }
6
7  ajax(CUSTOMER_API)({id:42})(renderCustomer);
8
9  var getCustomer = ajax(CUSTOMER_API);
10 var getCurrentUser = getCustomer({id:42});
11
12 getCustomer({id:42})(renderCustomer);
13
14 getCurrentUser(renderCustomer);
```

```
// var ajax = url => data => cb => { .. };
// var ajax = url => (data => (cb => { .. }));
var ajax = curry(
    3,
    function ajax(url,data,cb){ /*..*/ }
);
var getCustomer = ajax(CUSTOMER_API);
var getCurrentUser = getCustomer({id:42});
```

```
1  var ajax = curry(
2      3,
3      function ajax(url,data,cb){ /*..*/ }
4  );
5
6  // strict currying
7  ajax( CUSTOMER_API )( {id:42} )( renderCustomer );
8
9  // loose currying
10 ajax( CUSTOMER_API, {id:42} )( renderCustomer );
```

Partial Application vs Currying:

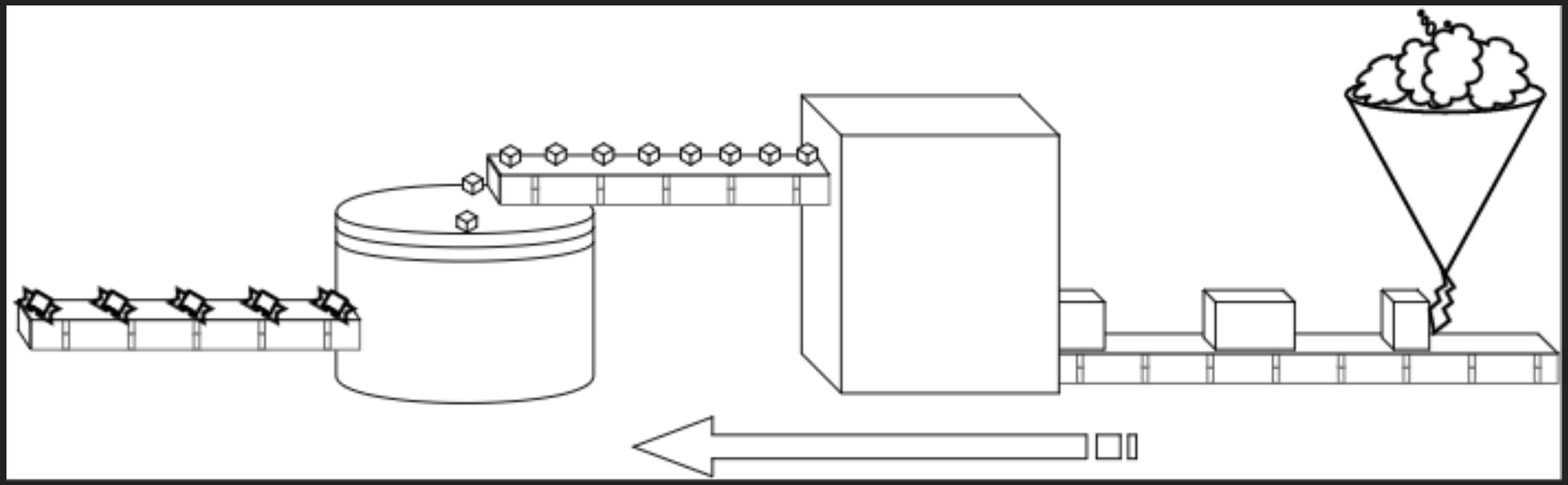
1. Both are specialization techniques
2. Partial Application presets some arguments now, receives the rest on the next call
3. Currying doesn't preset any arguments, receives each argument one at a time

Specialization Adapts Shape

```
1  function add(x,y) { return x + y; }
2
3  [0,2,4,6,8].map(function addOne(v){
4      return add(1,v);
5  });
6  // [1,3,5,7,9]
7
8  add = curry(add);
9
10 [0,2,4,6,8].map(add(1));
11 // [1,3,5,7,9]
```

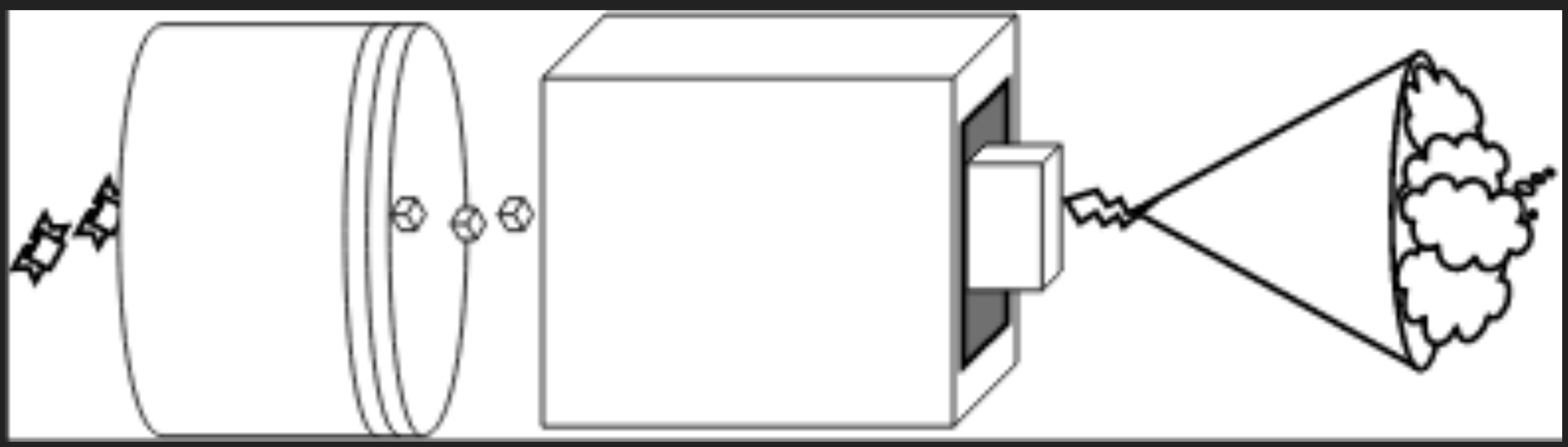
COMPOSITION

```
1 function minus2(x) { return x - 2; }
2 function triple(x) { return x * 3; }
3 function increment(x) { return x + 1; }
4
5 // add shipping rate
6 var tmp = increment(4);
7 tmp = triple(tmp);
8 totalCost = basePrice + minus2(tmp);
```

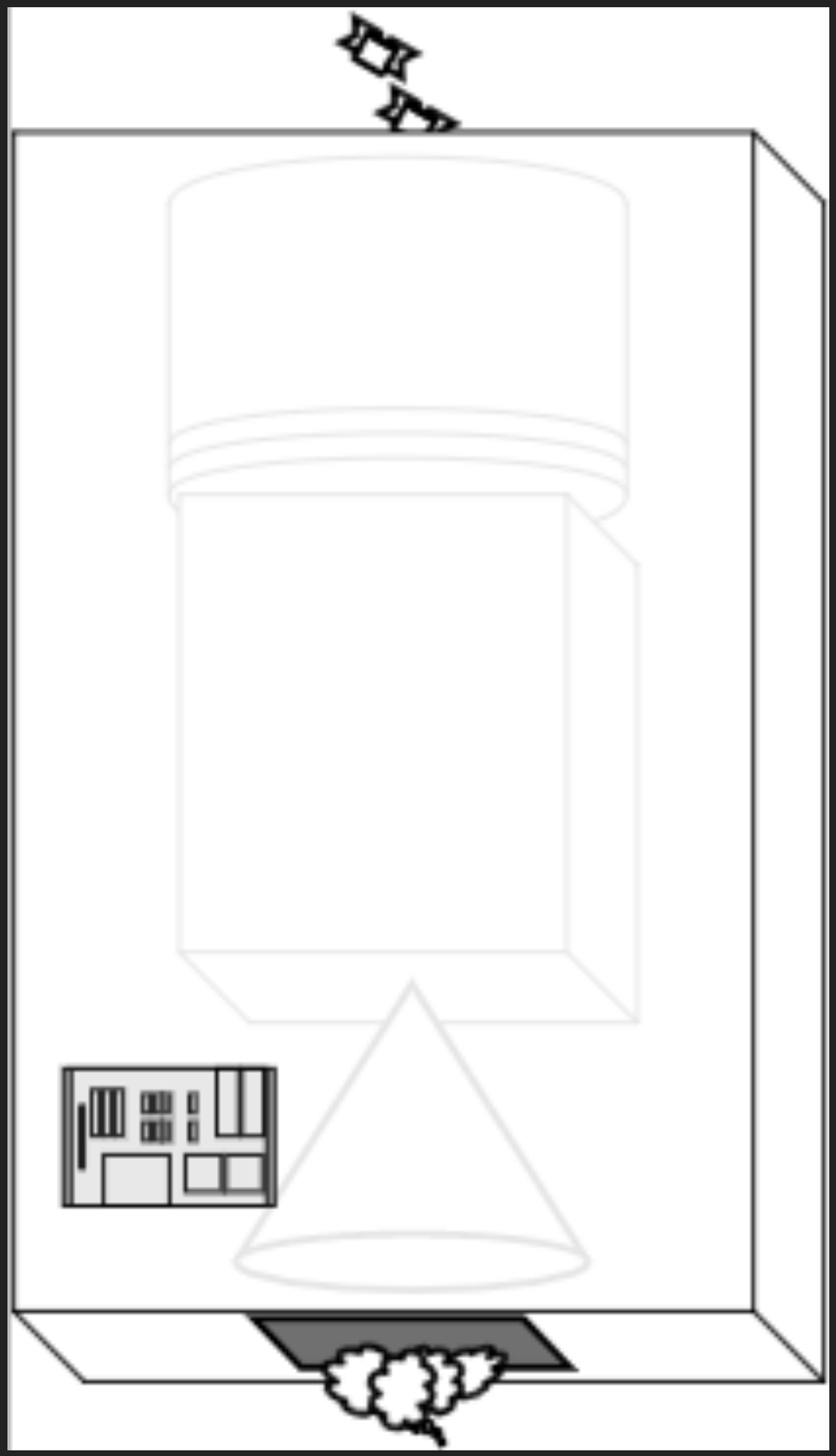


(RIGHT-TO-LEFT)

```
function minus2(x) { return x - 2; }
function triple(x) { return x * 3; }
function increment(x) { return x + 1; }

// add shipping rate
totalCost =
    basePrice +
    minus2(triple(increment(4)));
```

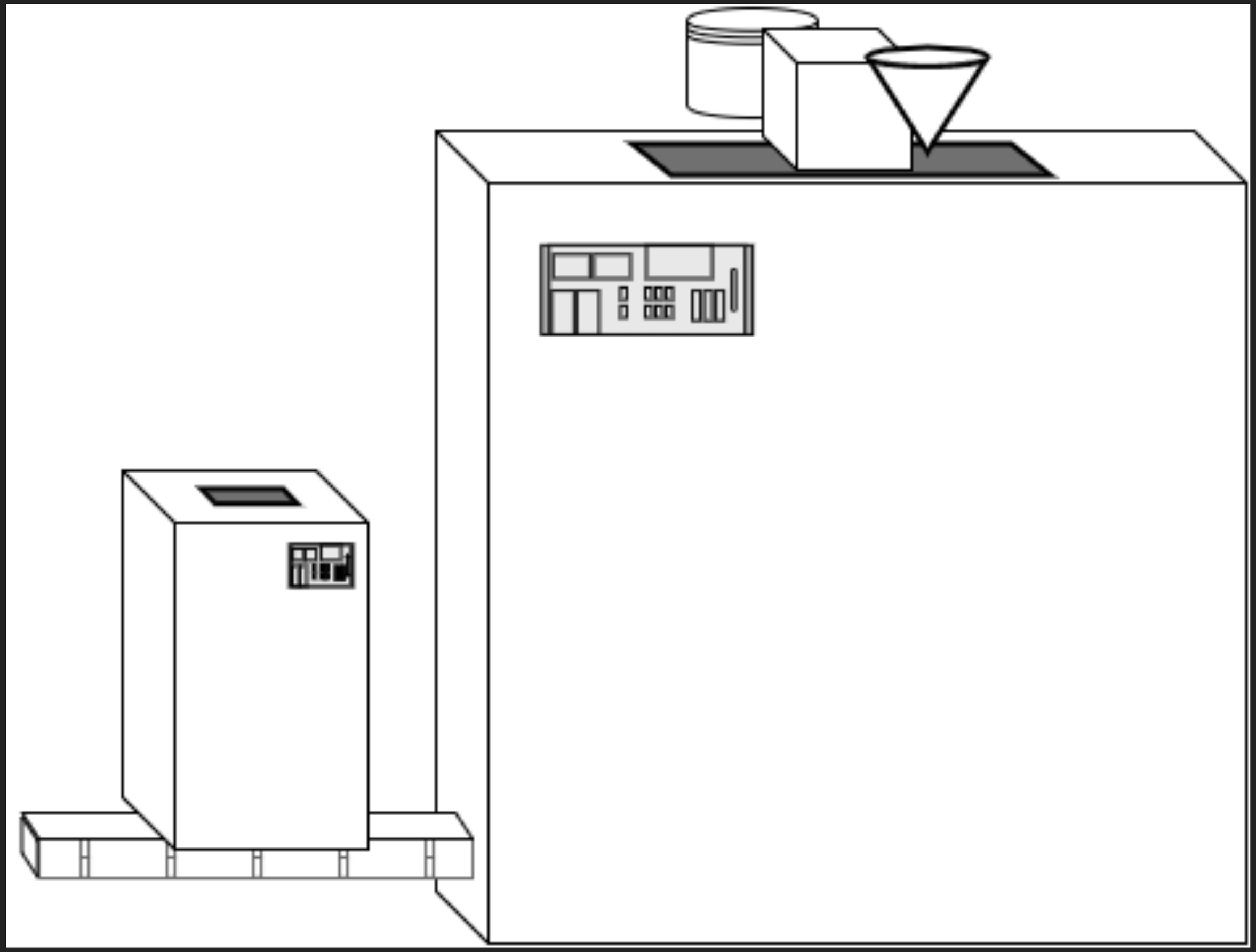


```
1  function minus2(x) { return x - 2; }
2  function triple(x) { return x * 3; }
3  function increment(x) { return x + 1; }
4
5  function shippingRate(x) {
6      return minus2(triple(increment(x)));
7  }
8
9  // add shipping rate
10 totalCost =
11     basePrice +
12     shippingRate(4);
```



```
1 function composeThree(fn3,fn2,fn1) {
2     return function composed(v){
3         return fn3(fn2(fn1(v)));
4     };
5 }
```

```
1 function minus2(x) { return x - 2; }
2 function triple(x) { return x * 3; }
3 function increment(x) { return x + 1; }
4
5 var shippingRate =
6     composeThree(minus2,triple,increment);
7
8 // calculate and add shipping rate
9 totalCost =
10     basePrice +
11     shippingRate(4);
```



```
1 function minus2(x) { return x - 2; }
2 function triple(x) { return x * 3; }
3 function increment(x) { return x + 1; }
4
5 var f = composeThree(minus2,triple,increment);
6 var p = composeThree(increment,triple,minus2);
7
8 f(4);   // 13
9 p(4);   // 7
10
11 var g = pipeThree(minus2,triple,increment);
12
13 g(4);   // 7
```

COMPOSE: RIGHT-TO-LEFT

PIPE: LEFT-TO-RIGHT

ASSOCIATIVITY

```
1  function minus2(x) { return x - 2; }
2  function triple(x) { return x * 3; }
3  function increment(x) { return x + 1; }
4
5  function composeTwo(fn2,fn1) {
6      return function composed(v){
7          return fn2(fn1(v));
8      };
9  }
10
11 var f = composeTwo(
12     composeTwo(minus2,triple),
13     increment
14 );
15 var p = composeTwo(
16     minus2,
17     composeTwo(triple,increment)
18 );
19
20 f(4);        // 13
21 p(4);        // 13
```

CURRYING REVISITED

```
1  function sum(x,y) { return x + y; }
2  function triple(x) { return x * 3; }
3  function divBy(y,x) { return x / y; }
4
5  divBy( 2, triple( sum(3,5) ) );  // 12
6
7  sum = curry(2,sum);
8  divBy = curry(2,divBy);
9
10 composeThree(
11     divBy(2),
12     triple,
13     sum(3)
14 )(5);          // 12
```

POINT-FREE REVISITED

```
1 var mod2 = mod(2);
2 var eq1 = eq(1);
3
4 // function isOdd(x) {
5 //   return x % 2 == 1;
6 // }
7 function isOdd(x) {
8     return eq1( mod2( x ) );
9 }
10
11 function composeTwo(fn2,fn1) {
12     return function composed(v){
13         return fn2( fn1( v ) );
14     };
15 }
16
17 var isOdd = composeTwo(eq1,mod2);
18 var isOdd = composeTwo(eq(1),mod(2));
```

IMMUTABILITY

ASSIGNMENT IMMUTABILITY

```
1  var basePrice = 89.99;
2  const shippingCost = 6.50;
3
4  // other code
5
6  basePrice += 5.00;        // allowed
7
8  // other code
9
10 shippingCost *= 1.04;    // not allowed!
```

```
1 var basePrice = 89.99;
2 const shippingCost = 6.50;
3
4 function increasePrice(price) {
5     return price + 5.00;
6 }
7 increasePrice(basePrice);   // 94.99
8
9 function increaseShipping(shipping) {
10     return shipping * 1.04;
11 }
12 increaseShipping(shippingCost);  // 6.76
```

```
1 {
2     const shippingCost = 6.50;
3     const updateOrder = compose(
4         saveOrderTotal,
5         computeOrderTotal(basePrice),
6         increaseShipping
7     );
8     updateOrder(shippingCost);
9 }
```

VALUE IMMUTABILITY

```
1  {
2      const orderDetails = {
3          orderId: 42,
4          total: (basePrice + shipping)
5      };
6
7      if (orderedItems.length > 0) {
8          orderDetails.items = orderedItems;
9      }
10
11     processOrder(orderDetails);
12 }
```

```
1  {
2      let orderDetails = {
3          orderId: 42,
4          total: (basePrice + shipping)
5      };
6
7      if (orderedItems.length > 0) {
8          orderDetails.items = orderedItems;
9      }
10
11     processOrder(Object.freeze(orderDetails));
12 }
```

Read-Only Data Structures: Data structures that never need to be mutated

```
1 function processOrder(order) {
2     if (!("status" in order)) {
3         order.status = "complete";
4     }
5
6     saveToDatabase(order);
7 }
```

```
1 function processOrder(order) {
2     var processedOrder = { ...order },
3     if (!("status" in order)) {
4         processedOrder.status = "complete";
5     }
6
7     saveToDatabase(processedOrder);
8 }
```

Treat all data structures as read-only whether they are or not

IMMUTABLE DATA STRUCTURES

```
1  var items = Immutable.List.of(
2      textbook,
3      supplies
4  );
5
6  var updatedItems = items.push(calculator);
7
8  items === updatedItems;          // false
9
10 items.size;                      // 2
11 updatedItems.size;               // 3
```

facebook.github.io/immutable-js

Immutable Data Structures: Data structures that need to be mutated

RECURSION

```
1  function isVowel(char) {
2      return ["a","e","i","o","u"].includes(char);
3  }
4
5  function countVowels(str) {
6      var count = 0;
7      for (var i = 0; i < str.length; i++) {
8          if (isVowel(str[i])) {
9              count++;
10         }
11     }
12     return count;
13 }
14
15 countVowels(
16     "The quick brown fox jumps over the lazy dog"
17 );
18 // 11
```

```
1  function countVowels(str) {
2      if (str.length == 0) return 0;
3      var first = (isVowel(str[0]) ? 1 : 0);
4      return first + countVowels( str.slice(1) );
5  }
6
7  countVowels(
8      "The quick brown fox jumps over the lazy dog"
9  );
10 // 11
```

```
1  function countVowels(str) {
2      var first = (isVowel(str[0]) ? 1 : 0);
3      if (str.length <= 1) return first;
4      return first + countVowels( str.slice(1) );
5  }
6
7  countVowels(
8      "The quick brown fox jumps over the lazy dog"
9  );
10 // 11
```

```
1  function countVowels(str) {
2      var first = (isVowel(str[0]) ? 1 : 0);
3      if (str.length <= 1) return first;
4      return first + countVowels( str.slice(1) );
5  }
6
7  countVowels(
8      "The quick brown fox jumps over the lazy dog"
9  );
10 // 11
```

PTC
PROPER TAIL CALLS

```
1 "use strict";
2
3 function decrement(x) {
4     return sub(x,1);
5 }
6
7 function sub(x,y) {
8     return x - y;
9 }
10
11 decrement(43);        // 42
```

```
1  "use strict";
2
3  function diminish(x) {
4      if (x > 90) {
5          return diminish(Math.trunc(x / 2));
6      }
7      return x - 3;
8  }
9
10 diminish(367);      // 42
```

```
1  function countVowels(str) {
2      var first = (isVowel(str[0]) ? 1 : 0);
3      if (str.length <= 1) return first;
4      return first + countVowels( str.slice(1) );
5  }
6
7  countVowels(
8      "The quick brown fox jumps over the lazy dog"
9  );
10 // 11
```

```
1  "use strict";
2
3  function countVowels(count,str) {
4      count += (isVowel(str[0]) ? 1 : 0);
5      if (str.length <= 1) return count;
6      return countVowels( count, str.slice(1) );
7  }
8
9  countVowels(
10     0,
11     "The quick brown fox jumps over the lazy dog"
12 );
13 // 11
```

```
1  "use strict";
2
3  var countVowels = curry(2,function countVowels(count,str){
4      count += (isVowel(str[0]) ? 1 : 0);
5      if (str.length <= 1) return count;
6      return countVowels( count, str.slice(1) );
7  })(0);
8
9  countVowels(
10     "The quick brown fox jumps over the lazy dog"
11 );
12 // 11
```

CPS

```
1  "use strict";
2
3  function countVowels(str,cont = v=>v) {
4      var first = (isVowel(str[0]) ? 1 : 0);
5      if (str.length <= 1) return cont(first);
6      return countVowels(str.slice(1),function f(v){
7          return cont(first + v);
8      });
9  }
10
11 countVowels(
12     "The quick brown fox jumps over the lazy dog"
13 );
14 // 11
```

TRAMPOLINES

```
1  function trampoline(fn) {
2      return function trampolined(...args) {
3          var result = fn(...args);
4
5          while (typeof result == "function") {
6              result = result();
7          }
8
9          return result;
10     };
11 }
```

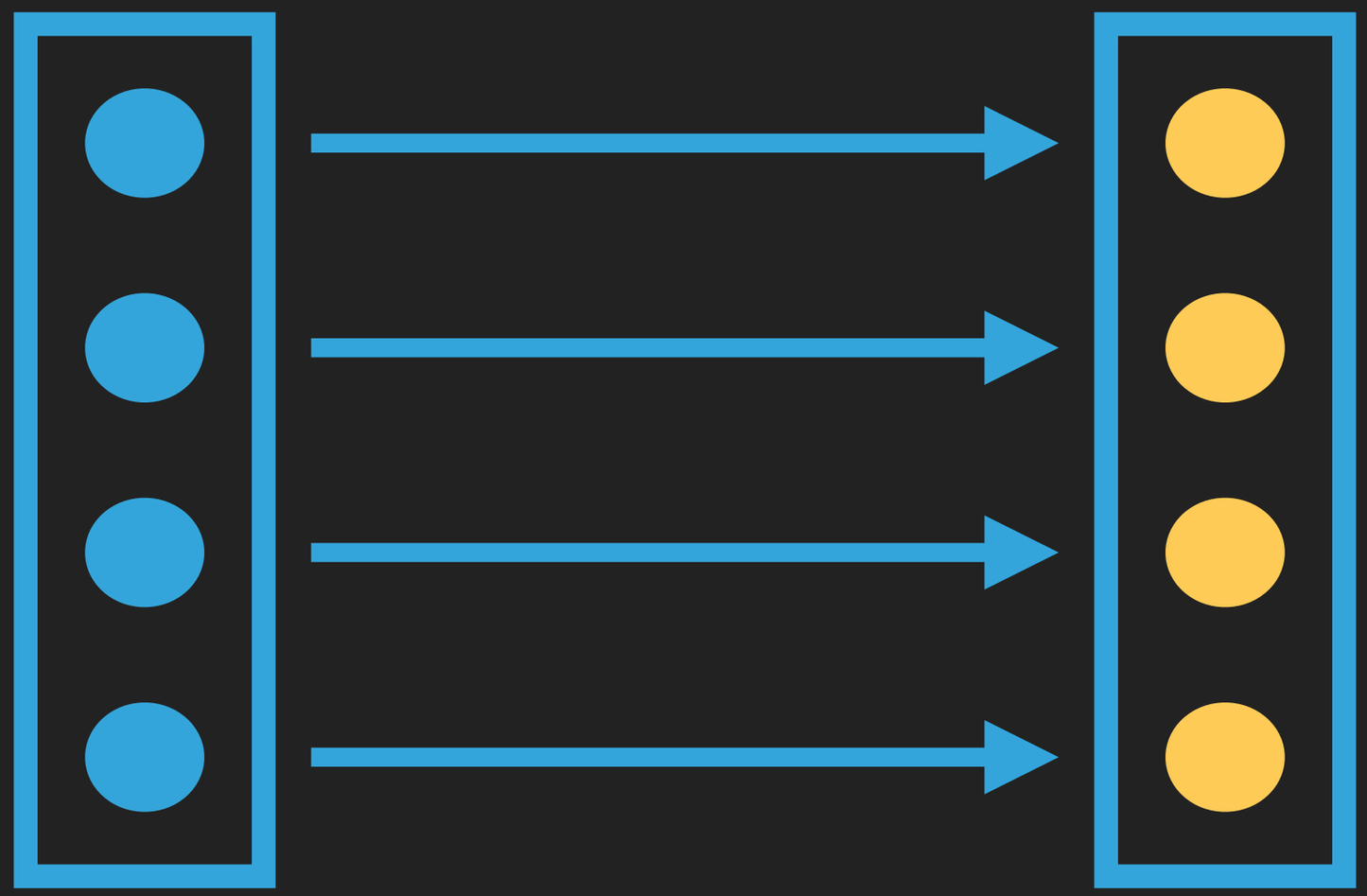
```
1  var countVowels =
2      trampoline(function countVowels(count,str){
3          count += (isVowel(str[0]) ? 1 : 0);
4          if (str.length <= 1) return count;
5          return function f(){
6              return countVowels(count,str.slice(1));
7          };
8      });
9
10 // optionally:
11 countVowels = curry(2,countVowels)(0);
```

```
3 function countVowels(count,str) {
4     count += (isVowel(str[0]) ? 1 : 0);
5     if (str.length <= 1) return count;
6     return countVowels( count, str.slice(1) );
7 }
```

```
1 function countVowels(count,str){
2     count += (isVowel(str[0]) ? 1 : 0);
3     if (str.length <= 1) return count;
4     return function f(){
5         return countVowels(count,str.slice(1));
6     };
7 }
```

LISTS

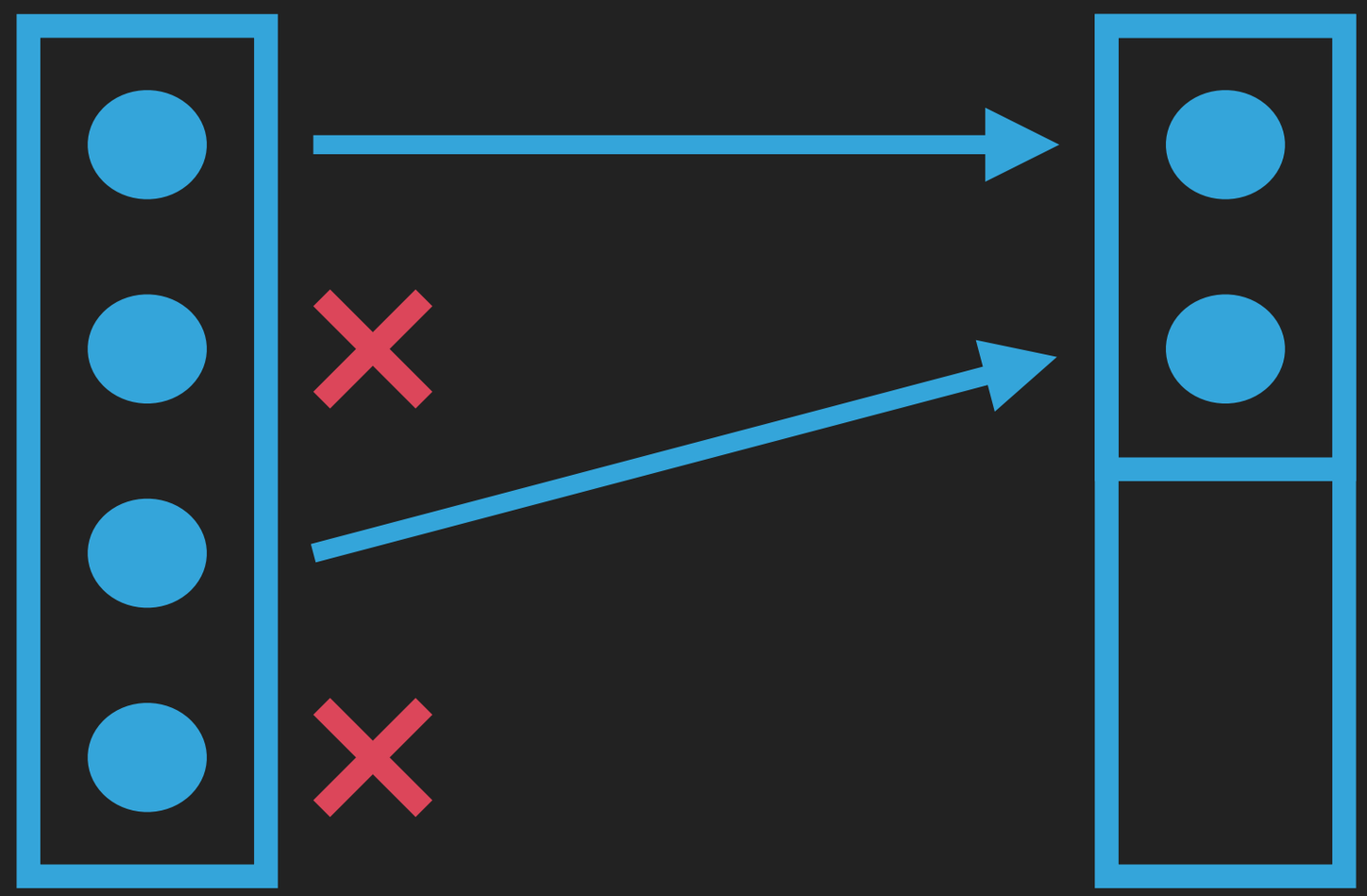
actually, data structures



MAP: TRANSFORMATION

```
1  function makeRecord(name) {
2      return { id: uniqID(), name };
3  }
4
5  function map(mapper,arr) {
6      var newList = [];
7      for (let elem of arr) {
8          newList.push( mapper(elem) );
9      }
10     return newList;
11 }
12
13 map(makeRecord,[ "Kyle", "Susan" ]);
14 // [ {id:42,name:"Kyle"}, {id:729,name:"Susan"} ]
```

```
1 function makeRecord(name) {
2     return { id: uniqID(), name };
3 }
4
5 [ "Kyle", "Susan" ].map(makeRecord);
6 // [ {id:42,name:"Kyle"}, {id:729,name:"Susan"} ]
```

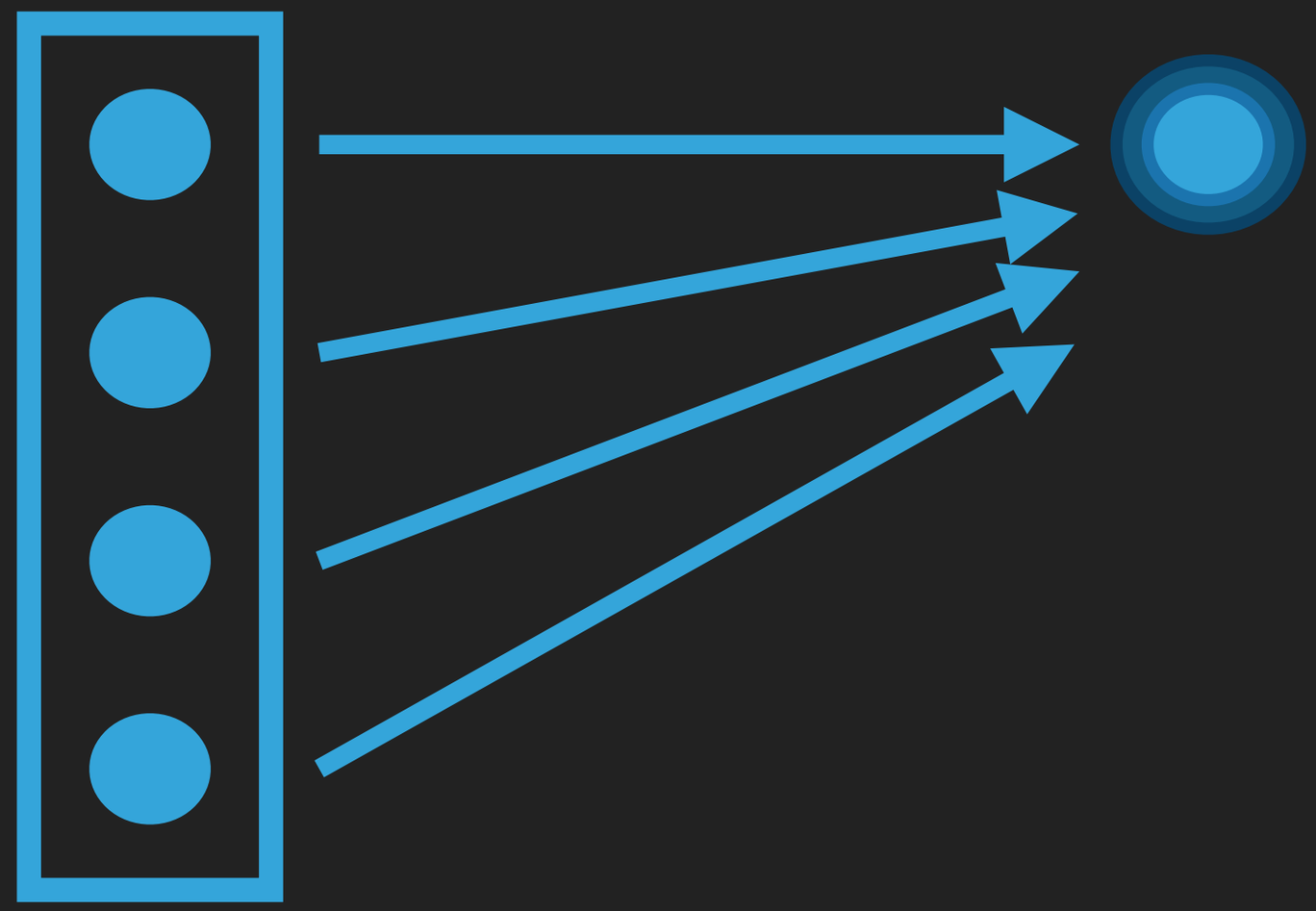


FILTER: EXCLUSION

ACTUALLY, INCLUSION?

```
1  function isLoggedIn(user) {
2      return user.session != null;
3  }
4
5  function filterIn(predicate,arr) {
6      var newList = [];
7      for (let elem of arr) {
8          if ( predicate(elem) ) {
9              newList.push(elem);
10         }
11     }
12     return newList;
13 }
14
15 filterIn(isLoggedIn,[
16     { userID: 42, session: "a%klDKF543_9*54" },
17     { userID: 17 },
18     { userID: 729, session: "HJ3434k$#.456" },
19 ]);
20 // [
21 //   { userID: 42, session: "a%klDKF543_9*54" },
22 //   { userID: 729, session: "HJ3434k$#.456" },
23 // ]
```

```
1  function isLoggedIn(user) {
2      return user.session != null;
3  }
4
5  [
6      { userID: 42, session: "a%klDKF543_9*54" },
7      { userID: 17 },
8      { userID: 729, session: "HJ3434k$#.456" },
9  ].filter(isLoggedIn);
10 // [
11 //   { userID: 42, session: "a%klDKF543_9*54" },
12 //   { userID: 729, session: "HJ3434k$#.456" },
13 // ]
```



REDUCE: COMBINING

```
1  function addToRecord(record,[key,value]) {
2      return { ...record, [key]: value };
3  }
4
5  function reduce(reducer,initialVal,arr) {
6      var ret = initialVal;
7      for (let elem of arr) {
8          ret = reducer(ret,elem);
9      }
10     return ret;
11 }
12
13 reduce(addToRecord,{},[
14     [ "name", "Kyle" ],
15     [ "age", 39 ],
16     [ "isTeacher", true ]
17 ]);
18 // { name: "Kyle", age: 39, isTeacher: true }
```

```
1  function addToRecord(record,[key,value]) {
2      return { ...record, [key]: value };
3  }
4
5  [
6      [ "name", "Kyle" ],
7      [ "age", 39 ],
8      [ "isTeacher", true ]
9  ].reduce(addToRecord,{});
10 // { name: "Kyle", age: 39, isTeacher: true }
```

COMPOSITION REVISITED

```
1  function add1(v) { return v + 1; }
2  function mul2(v) { return v * 2; }
3  function div3(v) { return v / 3; }
4
5  function composeTwo(fn2,fn1) {
6      return function composed(v){
7          return fn2(fn1(v));
8      };
9  }
10
11 var f = [div3,mul2,add1].reduce(composeTwo);
12 var p = [add1,mul2,div3].reduceRight(composeTwo);
13
14 f(8);        // 6
15 p(8);        // 6
```

```
1  function compose(...fns) {
2      return function composed(v){
3          return fns.reduceRight(function invoke(fn,val){
4              return fn(val);
5          },v);
6      };
7  }
8
9  var f = compose(div3,mul2,add1);
10
11 f(8);        // 6
```

FUSION

```
1 function add1(v) { return v + 1; }
2 function mul2(v) { return v * 2; }
3 function div3(v) { return v / 3; }
4
5 var list = [2,5,8,11,14,17,20];
6
7 list
8  .map( add1 )
9  .map( mul2 )
10 .map( div3 );
11 // [2,4,6,8,10,12,14]
```

```
1  function add1(v) { return v + 1; }
2  function mul2(v) { return v * 2; }
3  function div3(v) { return v / 3; }
4
5  var list = [2,5,8,11,14,17,20];
6
7  list
8  .map(
9      compose(div3,mul2,add1)
10 );
11 // [2,4,6,8,10,12,14]
```

TRANSDUCING

```
1  function add1(v) { return v + 1; }
2  function isOdd(v) { return v % 2 == 1; }
3  function sum(total,v) { return total + v; }
4
5  var list = [1,3,4,6,9,12,13,16,21];
6
7  list
8  .map( add1 )
9  .filter( isOdd )
10 .reduce( sum );
11 // 42
```

```
1  function add1(v) { return v + 1; }
2  function isOdd(v) { return v % 2 == 1; }
3  function sum(total,v) { return total + v; }
4
5  var list = [1,3,4,6,9,12,13,16,21];
6
7  list
8  .reduce(function allAtOnce(total,v){
9      v = add1(v);
10     if (isOdd(v)) {
11         total = sum(total,v);
12     }
13     return total;
14 },0);
15 // 42
```

```
1  function add1(v) { return v + 1; }
2  function isOdd(v) { return v % 2 == 1; }
3  function sum(total,v) { return total + v; }
4
5  var transducer = compose(
6      mapReducer(add1),
7      filterReducer(isOdd)
8  );
9
10 transduce(
11     transducer,
12     sum,
13     0,
14     [1,3,4,6,9,12,13,16,21]
15 );
16 // 42
17
18 into(transducer,0,[1,3,4,6,9,12,13,16,21]);
19 // 42
20
21 [1,3,4,6,9,12,13,16,21].reduce(transducer(sum),0);
22 // 42
```

DERIVING TRANSDUCTION

```
1 function add1(v) { return v + 1; }
2 function isOdd(v) { return v % 2 == 1; }
3 function sum(total,v) { return total + v; }
4
5 var list = [1,3,4,6,9,12,13,16,21];
6
7 list
8 .map( add1 )
9 .filter( isOdd )
10 .reduce( sum );
11 // 42
```

```
1  function mapWithReduce(arr,mappingFn) {
2      return arr.reduce(function reducer(list,v){
3          list.push( mappingFn(v) );
4          return list;
5      }, [] );
6  }
7
8  function filterWithReduce(arr,predicateFn) {
9      return arr.reduce(function reducer(list,v){
10         if (predicateFn(v)) list.push(v);
11         return list;
12     }, [] );
13 }
14
15 var list = [1,3,4,6,9,12,13,16,21];
16
17 list = mapWithReduce( list, add1 );
18 list = filterWithReduce( list, isOdd );
19 list.reduce( sum );
20 // 42
```

```
1  function mapReducer(mappingFn) {
2      return function reducer(list,v){
3          list.push( mappingFn(v) );
4          return list;
5      };
6  }
7
8  function filterReducer(predicateFn) {
9      return function reducer(list,v){
10         if (predicateFn(v)) list.push(v);
11         return list;
12     };
13 }
14
15 var list = [1,3,4,6,9,12,13,16,21];
16
17 list
18 .reduce( mapReducer(add1), [] )
19 .reduce( filterReducer(isOdd), [] )
20 .reduce( sum );
21 // 42
```

```
1  function listCombination(list,v) {
2      list.push(v);
3      return list;
4  }
5
6  function mapReducer(mappingFn) {
7      return function reducer(list,v){
8          return listCombination( list, mappingFn(v) );
9      };
10 }
11
12 function filterReducer(predicateFn) {
13     return function reducer(list,v){
14         if (predicateFn(v)) return listCombination( list, v );
15         return list;
16     };
17 }
18
19 var list = [1,3,4,6,9,12,13,16,21];
20
21 list
22 .reduce( mapReducer(add1), [] )
23 .reduce( filterReducer(isOdd), [] )
24 .reduce( sum );
25 // 42
```

```
1  function listCombination(list,v) {
2      list.push(v);
3      return list;
4  }
5
6  var mapReducer = curry(2,function mapReducer(mappingFn,combineFn){
7      return function reducer(list,v){
8          return combineFn( list, mappingFn(v) );
9      };
10 });
11
12 var filterReducer = curry(2,function filterReducer(predicateFn,combineFn){
13     return function reducer(list,v){
14         if (predicateFn(v)) return combineFn( list, v );
15         return list;
16     };
17 });
18
19 var list = [1,3,4,6,9,12,13,16,21];
20
21 list
22 .reduce( mapReducer(add1)(listCombination), [] )
23 .reduce( filterReducer(isOdd)(listCombination), [] )
24 .reduce( sum );
25 // 42
```

```
1  function listCombination(list,v) {
2      list.push(v);
3      return list;
4  }
5
6  var mapReducer = curry(2,function mapReducer(mappingFn,combineFn){
7      return function reducer(list,v){
8          return combineFn( list, mappingFn(v) );
9      };
10 });
11
12 var filterReducer = curry(2,function filterReducer(predicateFn,combineFn){
13     return function reducer(list,v){
14         if (predicateFn(v)) return combineFn( list, v );
15         return list;
16     };
17 });
18
19 var transducer = compose( mapReducer(add1), filterReducer(isOdd) );
20
21 var list = [1,3,4,6,9,12,13,16,21];
22
23 list
24 .reduce( transducer(listCombination), [] )
25 .reduce( sum );
26 // 42
```

```javascript
var mapReducer = curry(2,function mapReducer(mappingFn,combineFn){
    return function reducer(list,v){
        return combineFn( list, mappingFn(v) );
    };
});

var filterReducer = curry(2,function filterReducer(predicateFn,combineFn){
    return function reducer(list,v){
        if (predicateFn(v)) return combineFn( list, v );
        return list;
    };
});

var transducer = compose( mapReducer(add1), filterReducer(isOdd) );

var list = [1,3,4,6,9,12,13,16,21];

list
.reduce( transducer(sum), 0 );
// 42
```

```
function add1(v) { return v + 1; }
function isOdd(v) { return v % 2 == 1; }
function sum(total,v) { return total + v; }

var transducer = compose(
    mapReducer(add1),
    filterReducer(isOdd)
);

transduce(
    transducer,
    sum,
    0,
    [1,3,4,6,9,12,13,16,21]
);
// 42

into(transducer,0,[1,3,4,6,9,12,13,16,21]);
// 42
```

DATA STRUCTURE OPERATIONS

```
1  var obj = {
2      name: "Kyle",
3      email: "Getify@Gmail.com"
4  };
5
6  function mapObj(mapper,o) {
7      var newObj = {};
8      for (let key of Object.keys(o)) {
9          newObj[key] = mapper( o[key] );
10     }
11     return newObj;
12 }
13
14 mapObj(function lower(val){
15     return val.toLowerCase();
16 },obj);
17 // { name: "kyle", email: "getify@gmail.com" }
```

MONAD:
FP DATA STRUCTURE

~~Monad: a monoid in the category of endofunctors~~

Monad: a pattern for pairing data with a set of predictable behaviors that let it interact with other data+behavior pairings (monads)

```
1  function Just(val) {
2      return { map, chain, ap };
3
4      // **********************
5
6      function map(fn) {
7          return Just( fn( val ) );
8      }
9
10     // aka: bind, flatMap
11     function chain(fn) {
12         return fn( val );
13     }
14
15     function ap(anotherMonad) {
16         return anotherMonad.map( val );
17     }
18 }
```

```
1  var fortyOne = Just(41);
2  var fortyTwo = fortyOne.map(function inc(v){
3      return v + 1;
4  });
5
6  function identity(v) {
7      return v;
8  }
9
10 // debug inspection:
11 fortyOne.chain(identity);    // 41
12 fortyTwo.chain(identity);    // 42
```

```
1 var user1 = Just("Kyle");
2 var user2 = Just("Susan");
3
4 var tuple = curry(2,function tuple(x,y){
5     return [x,y];
6 });
7
8 var users = user1.map(tuple).ap(user2);
9
10 // debug inspection:
11 users.chain(identity);  // ["Kyle","Susan"]
```

```
1  var someObj = { something: { else: { entirely: 42 } } };
2  // someObj.something.else.entirely;   // 42
3
4  function Nothing() {
5      return { map: Nothing, chain: Nothing, ap: Nothing };
6  }
7
8  var Maybe = { Just, Nothing, of: Just };
9
10 function fromNullable(val) {
11     if (val == null) return Maybe.Nothing();
12     else return Maybe.of(val);
13 }
14
15 var prop = curry(2, function prop(prop,obj){
16     return fromNullable( obj[prop] );
17 });
18
19 Maybe.of( someObj )
20 .chain( prop( "something" ) )
21 .chain( prop( "else" ) )
22 .chain( prop( "entirely" ) )
23
24 // debug inspection:
25 .chain( identity );              // 42
```

There are many kinds of monads:
Just, Nothing, Maybe, Either, IO, etc

Should you use monads?
Maybe.

ASYNC

```
1 var a = [1,2,3];
2
3 var b = a.map(function double(v){
4         return v * 2;
5 });
6
7 b;                // [2,4,6]
```

SYNCHRONOUS, EAGER FP

LAZY FP, OVER TIME?

```
1  var a = [];
2
3  var b = mapLazy(function double(v){
4      return v * 2;
5  },a);
6
7  a.push(1);
8
9  a[0];          // 1
10 b[0];          // 2
11
12 a.push(2);
13
14 a[1];          // 2
15 b[1];          // 4
```

```
1  var a = new LazyArray();
2
3  setInterval(function everySecond(){
4      a.push(Math.random());
5  },1000);
6
7  // ******************************
8
9  var b = a.map(function double(v){
10     return v * 2;
11 });
12
13 b.forEach(function onValue(v){
14     console.log(v);
15 });
```

"LAZYARRAY"

≅

OBSERVABLE

```
1  var a = new Rx.Subject();
2
3  setInterval(function everySecond(){
4      a.next(Math.random());
5  },1000);
6
7  // ***************************
8
9  var b = a.map(function double(v){
10     return v * 2;
11 });
12
13 b.subscribe(function onValue(v){
14     console.log(v);
15 });
```

FP LIBRARIES

LODASH/FP

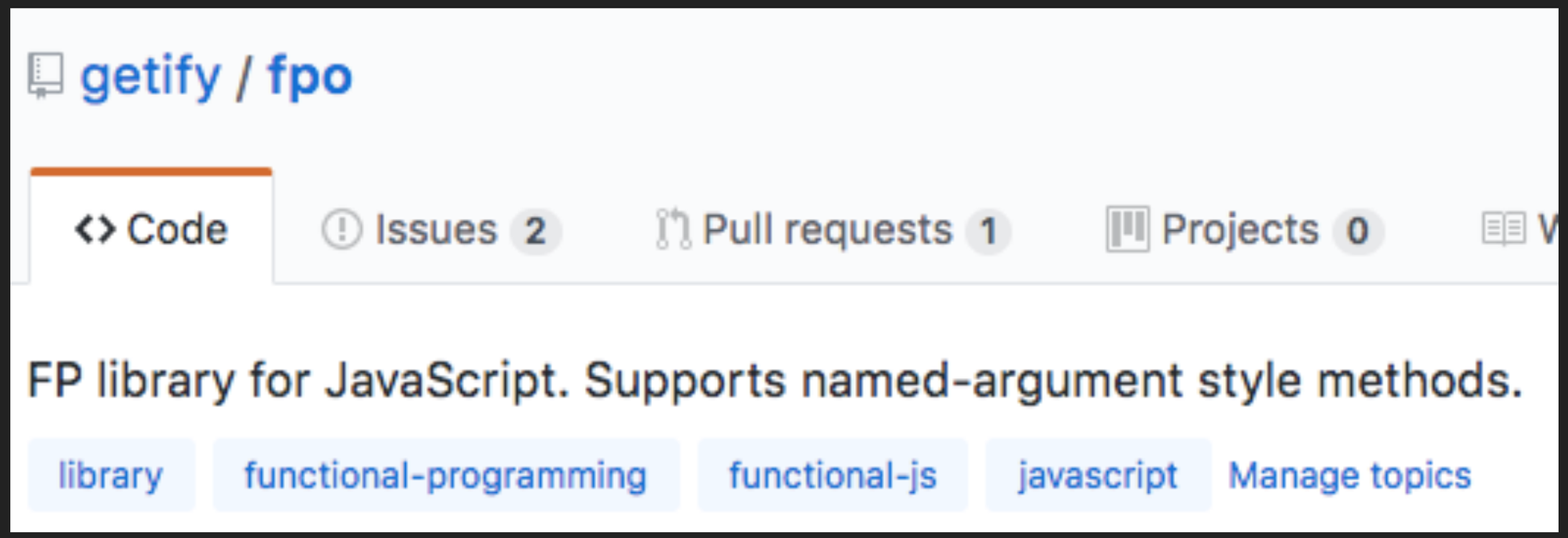
github.com/lodash/lodash/wiki/FP-Guide

```
1  // var fp = require("lodash/fp");
2  fp.reduce(
3      (acc,v) => acc + v,
4      0,
5      [3,7,9]
6  );
7  // 19
8
9  var f = fp.curryN(3,function f(x,y,z){
10     return x + (y * z);
11 });
12 var g = fp.compose([
13     fp.add(1),
14     f(1,4)
15 ]);
16
17 g(10);
18 // 42
```

RAMDA

ramdajs.com

```
1 R.reduce(
2         (acc,v) => acc + v,
3         0,
4         [3,7,9]
5 );
6 // 19
7
8 var f = R.curryN(3,function f(x,y,z){
9         return x + (y * z);
10 });
11 var g = R.compose(
12         R.inc,
13         f(1,4)
14 );
15
16 g(10);
17 // 42
```



github.com/getify/fpo

```
// the classic/traditional method style
// (on the `FPO.std.*` namespace)
FPO.std.reduce(
    (acc,v) => acc + v,
    undefined,
    [3,7,9]
);  // 19

// FPO named-argument method style
FPO.reduce({
    arr: [3,7,9],
    fn: ({acc,v}) => acc + v
}); // 19
```

```
1 var f = curry(
2         2,
3         flip(partialRight(reduce,[[3,7,9]]))
4 )(0);
5
6 f((acc,v) => acc + v);  // 19
7 f((acc,v) => acc * v);  // 189
```

```
1 var f = FPO.reduce({ arr: [3,7,9] });
2
3 // later:
4 f({ fn: ({acc,v}) => acc + v });  // 19
5 f({ fn: ({acc,v}) => acc * v });  // 189
```

RECAP:

- Functions (~~side effects~~, point-free)
- Closure
- Composition
- Immutability
- Recursion
- Lists & Data Structures
- Async (observables)

THANKS!!!!

KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS