

KYLE SIMPSON

GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS

 **getify / Functional-Light-JS**

 Code

 Issues **28**

 Pull requests **3**

 Projects **0**

A book about functional programming in JavaScript.

book

javascript

functional-programming

training-materials

trainin

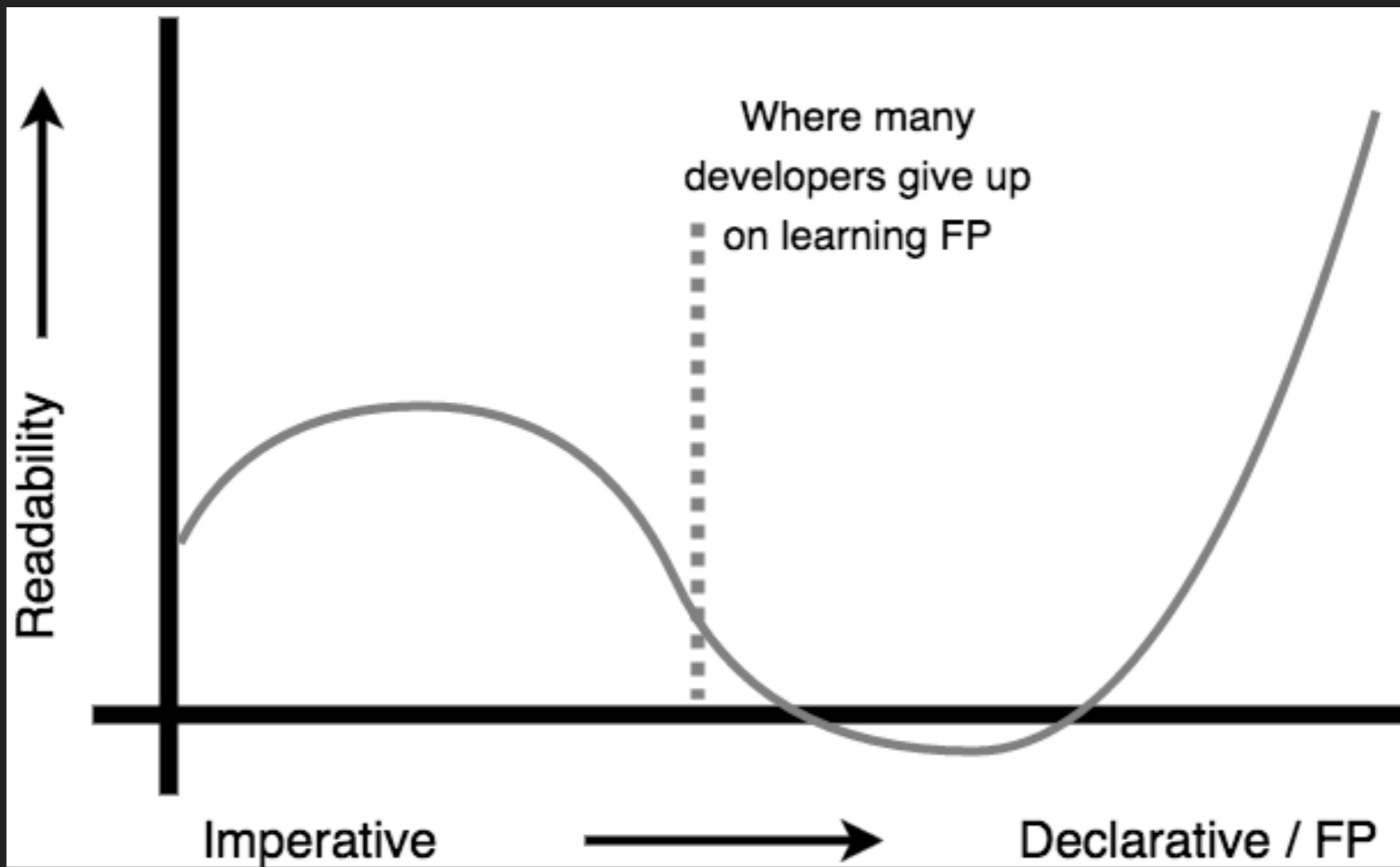
github.com/getify/Functional-Light-JS

WHY FP?

IMPERATIVE

VS

DECLARATIVE



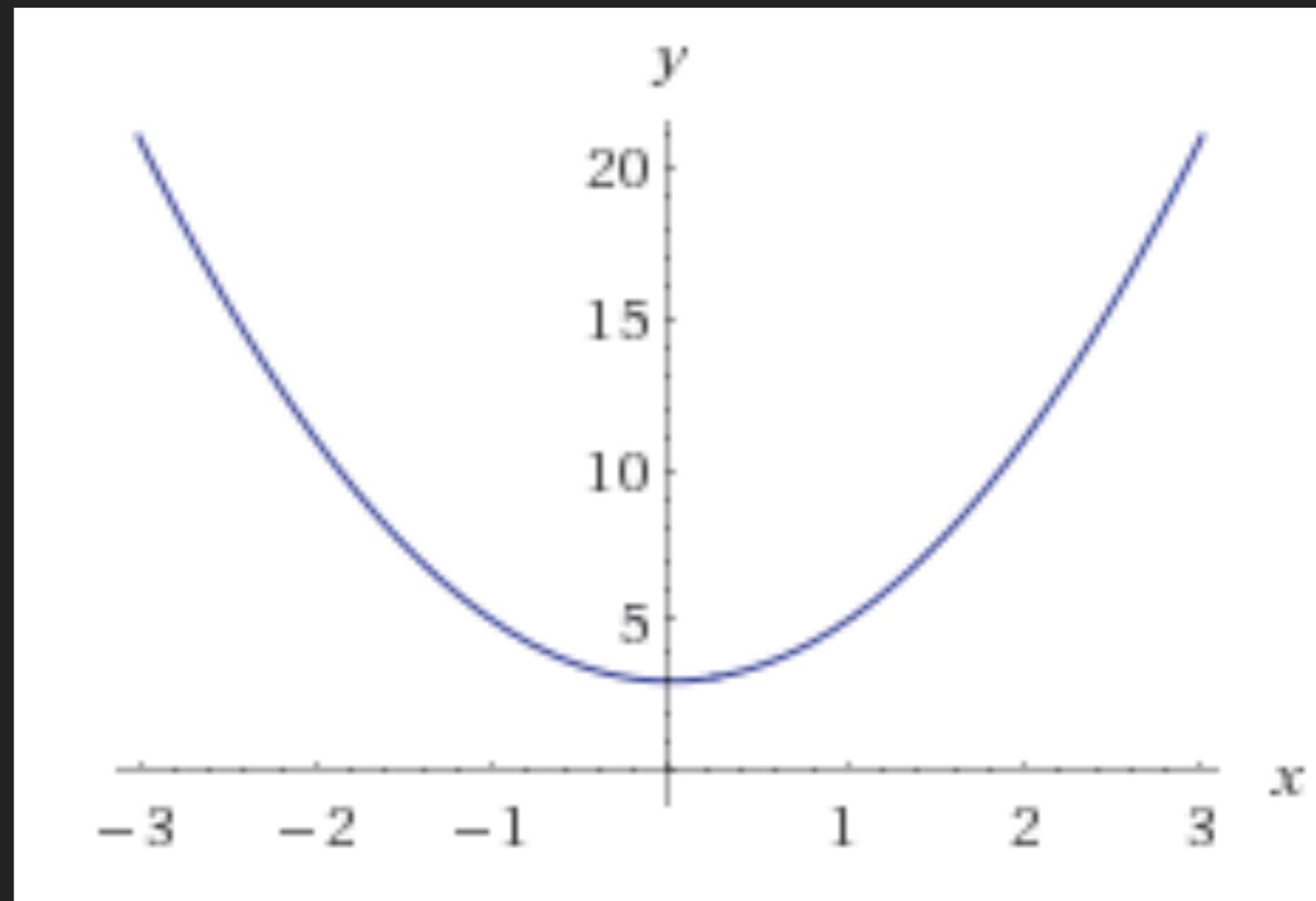
PROVABLE

LESS TO READ

FUNCTIONS

```
1 function foo(x,y,z,w) {
2     console.log( x, y, z, w );
3 }
4
5 function bar(x = 2, ...args) {
6     return foo(x,42,...args);
7 }
8
9
10 bar();           // 2 42 undefined undefined
11
12 bar(3,8,11);     // 3 42 8 11
13
14 bar(...[6,5]);  // 6 42 5 undefined
```

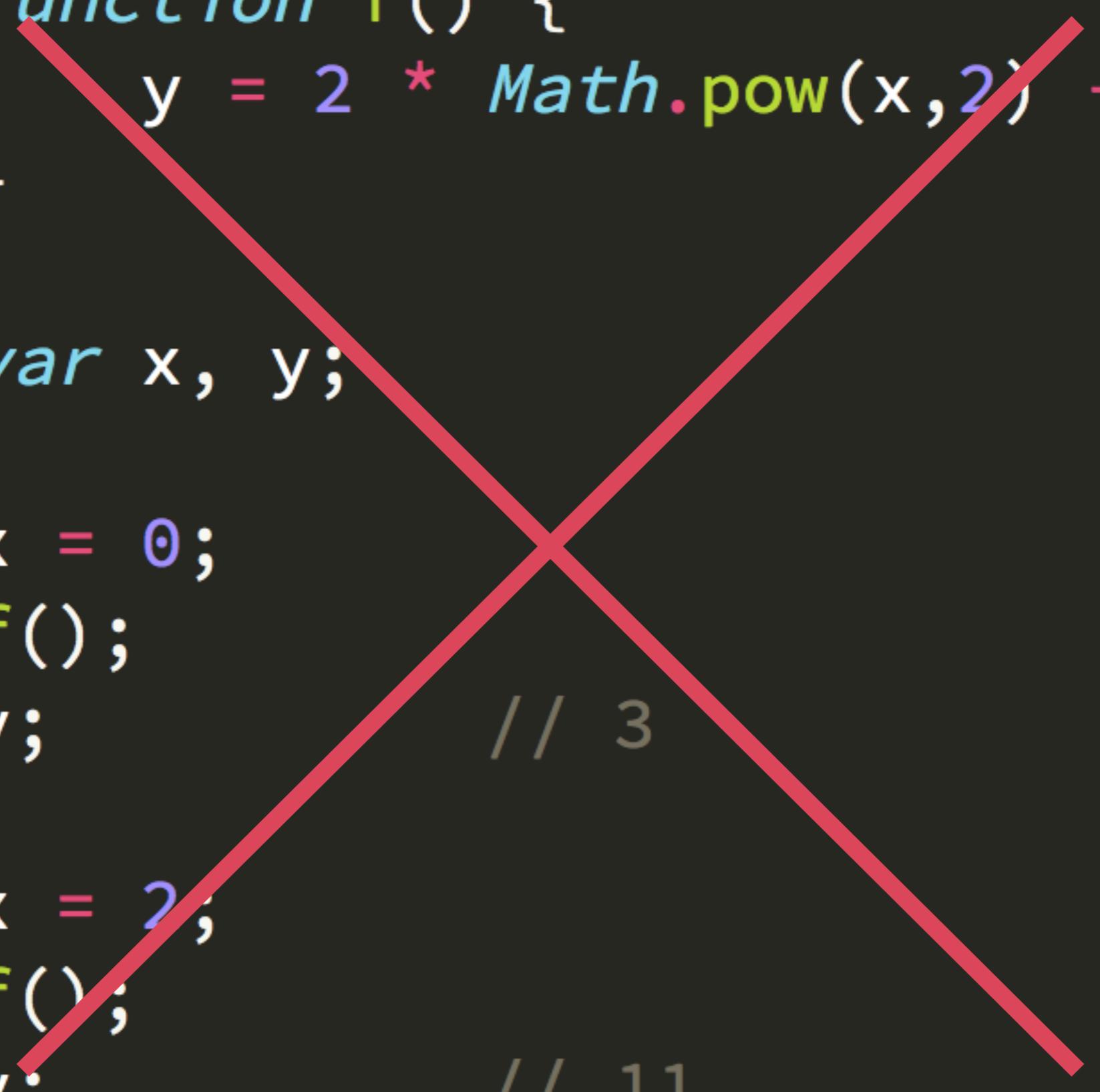
```
1 function foo(x,y) {  
2     return [x + 1, y - 1];  
3 }  
4  
5 var [a,b] = foo(...[5,10]);  
6  
7 a;           // 6  
8 b;           // 9
```



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

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

```
1 function f() {  
2     y = 2 * Math.pow(x,2) + 3;  
3 }  
4  
5 var x, y;  
6  
7 x = 0;  
8 f();  
9 y; // 3  
10  
11 x = 2;  
12 f();  
13 y; // 11
```



~~SIDE EFFECTS~~

```
1 function f(x) {  
2     return 2 * Math.pow(x,2) + 3;  
3 }  
4  
5 var y;  
6  
7 y = f(0);  
8 // 3  
9  
10 y = f(2);  
11 // 11  
12  
13 y = f(-1);  
14 // 5
```

PURE FUNCTIONS

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

PURIFYING

```
1 function f() {
2     y = 2 * Math.pow(x,2) + 3;
3 }
4
5 var x, y;
6
7 x = 0;
8 f();
9 y;           // 3
10
11 x = 2;
12 f();
13 y;         // 11
```

```
1  function F(x) {
2      var y;
3      f(x);
4      return y;
5
6      function f() {
7          y = 2 * Math.pow(x,2) + 3;
8      }
9  }
10
11 var y;
12
13 y = F(0);
14 // 3
15
16 y = F(2);
17 // 11
```

```
1  function f() {
2      y = 2 * Math.pow(x,2) + 3;
3  }
4
5  function F(curX) {
6      var [origX,origY] = [x,y];
7      x = curX;
8      f();
9      var newY = y;
10     [x,y] = [origX,origY];
11     return newY;
12 }
13
14 var x, y;
15
16 F(0);
17 // 3
18
19 F(2);
20 // 11
```

EXERCISE 1

```
1 const y = 1;
2
3 function foo(x) {
4     return x + y;
5 }
6
7 foo(1); // 2
```

```
1 function foo(x) {  
2     return bar(x);  
3 }  
4  
5 function bar(y) {  
6     return y + 1;  
7 }  
8  
9 foo(1); // 2
```

```
1 function foo(bar) {
2     return function(x) {
3         return bar(x);
4     };
5 }
6
7 foo(function(v) {
8     return v * 2;
9 }) (3);
10 // 6
```

```
1 function getId(obj) {  
2     return obj.id;  
3 }
```

```
4  
5 getId({  
6     get id() {  
7         return Math.random();  
8     }  
9 });
```

ARGUMENTS

```
1 // unary
2 function increment(x) {
3     return sum(x,1);
4 }
5
6 // binary
7 function sum(x,y) {
8     return x + y;
9 }
```

```
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     console.log(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     console.log(args);
9 }
10
11 var g = flip(f);
12
13 g(1, 2, 3, 4);    // [2, 1, 3, 4]
```

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

POINT-FREE

```
1 foo ( function (v) {  
2     return bar (v) ;  
3 } ) ;  
4  
5 foo (bar) ;  
6
```

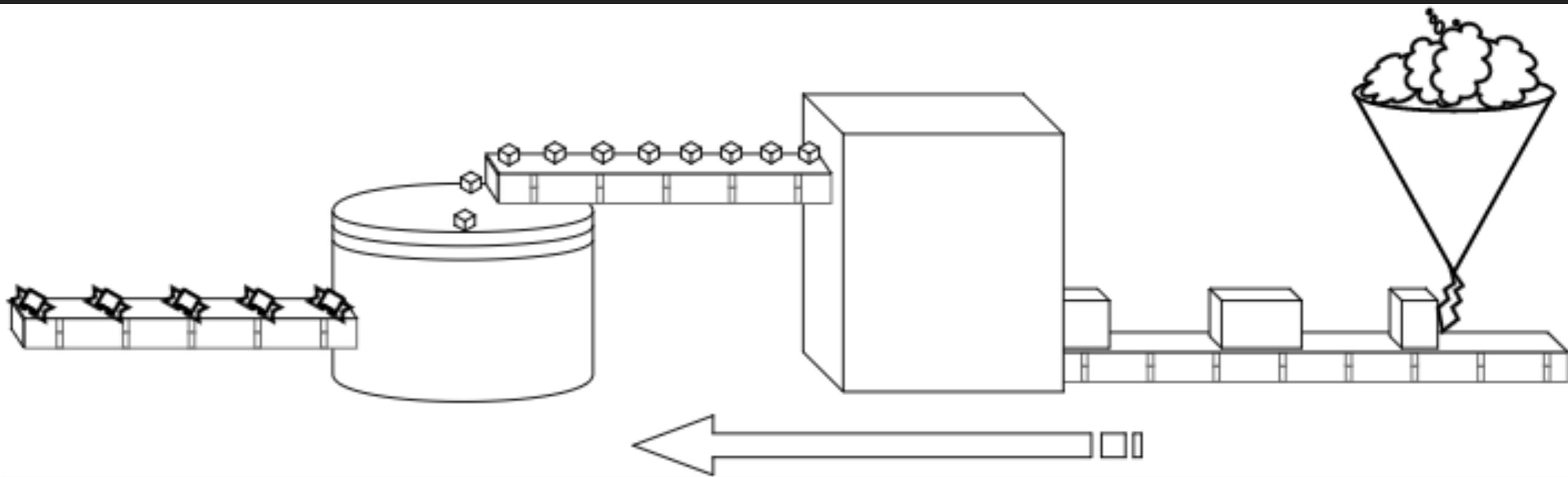
```
1 function isOdd(v) {
2     return v % 2 == 1;
3 }
4
5 function isEven(v) {
6     return !isOdd(v);
7 }
8
9 isEven(4); // true
```

```
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
```

EXERCISE 2

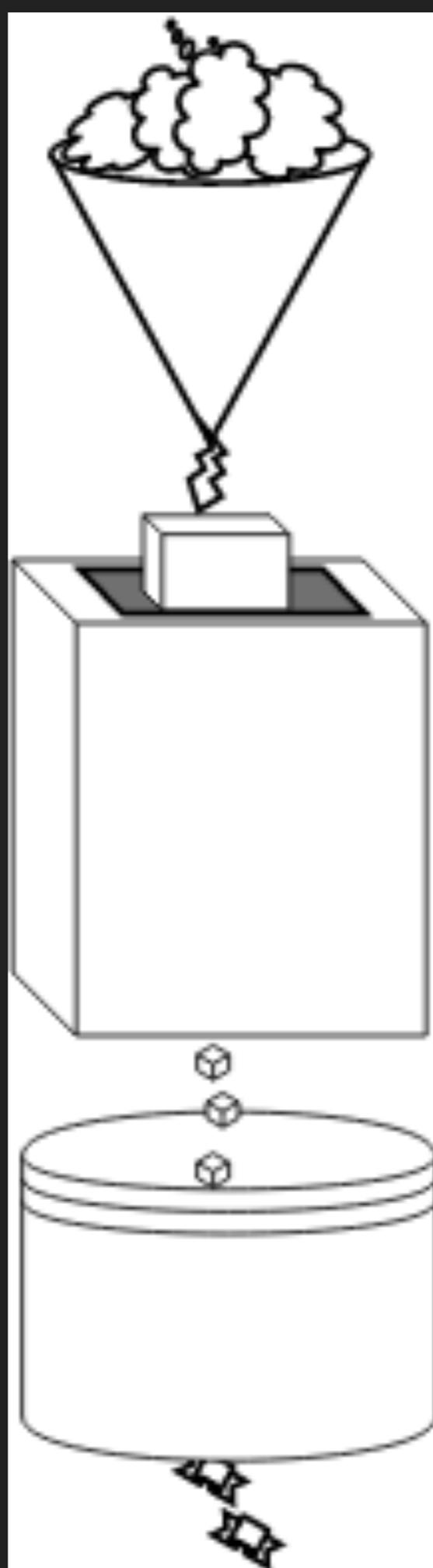
COMPOSITION

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

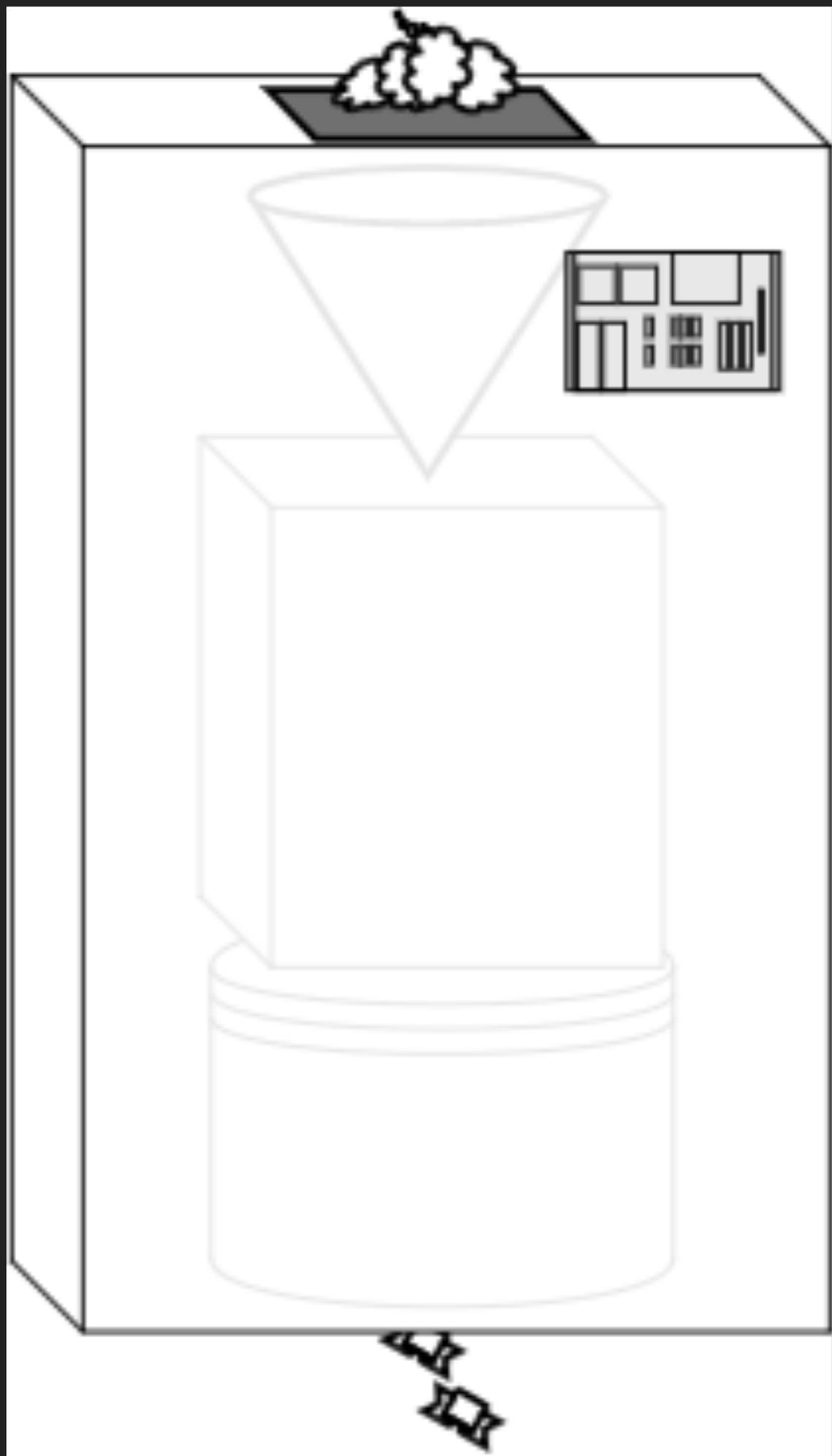


(RIGHT-TO-LEFT)

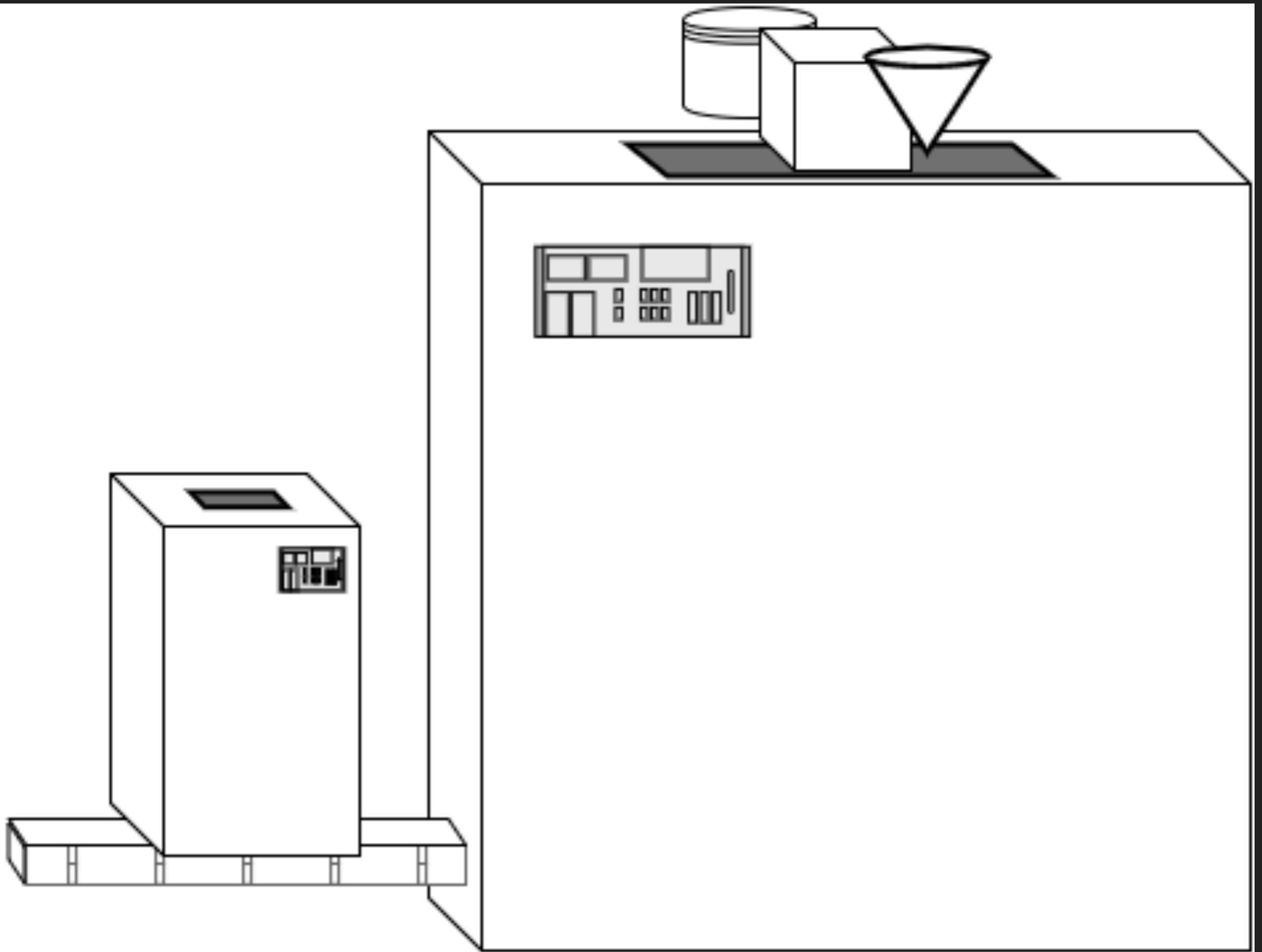
```
1 function sum(x,y) {
2     return x + y;
3 }
4
5 function mult(x,y) {
6     return x * y;
7 }
8
9 // (3 * 4) + 5
10 sum( mult( 3, 4 ), 5 ); // 17
```



```
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 // (3 * 4) + 5
14 multAndSum(3,4,5); // 17
```



```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  function pipe2(fn1,fn2) {
10     return function piped(arg1,arg2,arg3) {
11         return fn2(
12             fn1(arg1,arg2),
13             arg3
14         );
15     };
16 }
17
18 var multAndSum = pipe2(mult,sum);
19
20 // (3 * 4) + 5
21 multAndSum(3,4,5);           // 17
```



```
1 foo(bar(baz(2)));  
2  
3 compose(foo, bar, baz)(2);  
4  
5 pipe(baz, bar, foo)(2);
```

(RIGHT-TO-LEFT)

(LEFT-TO-RIGHT)

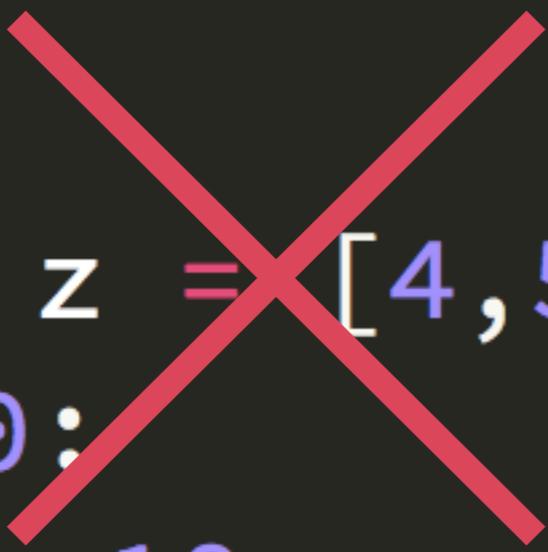
```
1 function composeRight(fn2, fn1) {  
2     return function comp(...args) {  
3         return fn2(fn1(...args));  
4     };  
5 }
```

```
1  function increment(x) {
2      return x + 1;
3  }
4
5  function double(x) {
6      return x * 2;
7  }
8
9  var f = composeRight(increment, double);
10 var p = composeRight(double, increment);
11
12 f(3);    // 7
13 p(3);    // 8
```

EXERCISE 3

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!
```



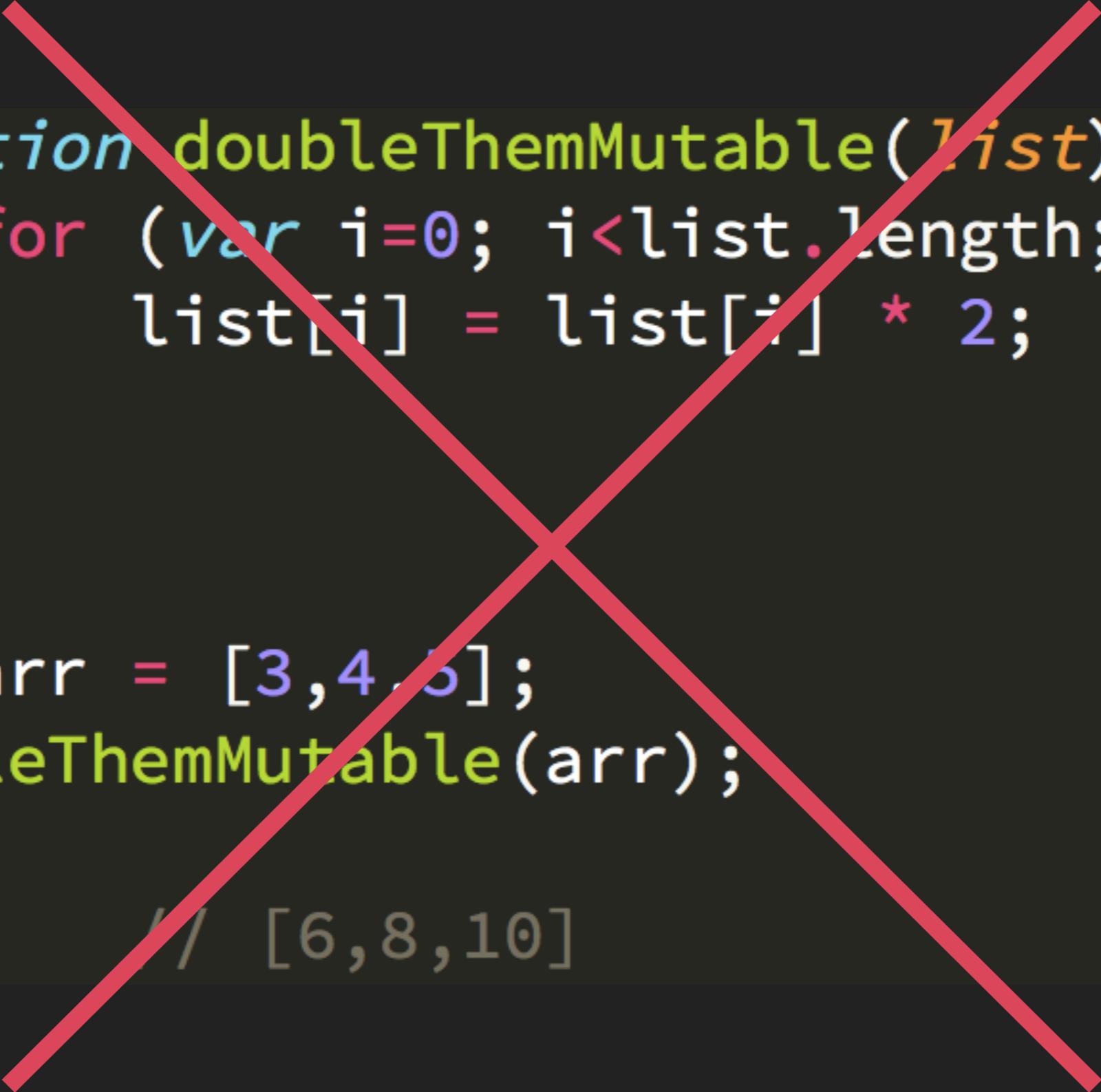
```
1 {  
2   const y = 3;  
3   var x = x + y;  
4   var z = foo(x);  
5   bar(x, y, z);  
6   z = [x*2, y*3, z*4];  
7   bam(z);  
8 }
```

```
1 var z = Object.freeze([4,5,6,[7,8,9]]);
2
3 z[0] = 10; // not allowed
4 z[3][0] = 10; // allowed!
```

```
1 var state = Immutable.List.of( 1, 2, 3, 4 );
2 var newState = state.set( 42, "meaning of life" );
3
4 state === newState;           // false
5
6 state.get( 2 );               // 3
7 newState.get( 2 );           // 3
8
9 state.get( 42 );              // undefined
10 newState.get( 42 );          // "meaning of life"
11
12 newState.toArray().slice( 1, 3 ); // [2,3]
```

facebook.github.io/immutable-js

```
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]
```



```
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]
```

EXERCISE 4

CLOSURE

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

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

```
1 function composeRight(fn2, fn1) {  
2     return function comp(...args) {  
3         return fn2(fn1(...args));  
4     };  
5 }
```

EXERCISE 5

**GENERALIZED
TO SPECIALIZED**

```
1 function add(x,y) {  
2     return x + y;  
3 }
```

```
4  
5 function partial(fn,...firstArgs) {  
6     return function applied(...lastArgs){  
7         return fn(...firstArgs,...lastArgs);  
8     };  
9 }
```

```
10  
11 var addTo10 = partial(add,10);  
12  
13 addTo10(32); // 42
```

```
1 var add3 = curry(function add3(x,y,z) {
2     return x + y + z;
3 });
4
5 var f = add3(3);
6
7 var p = f(4);
8
9 p(5);           // 12
10
11 add3(3)(4)(5); // 12
```


RECURSION

```
1 function sumIter(...nums) {
2     var sum = 0;
3     for (var i=0; i<nums.length; i++) {
4         sum = sum + nums[i];
5     }
6     return sum;
7 }
8
9 sumIter(3,4,5,6,7,8,9); // 42
```

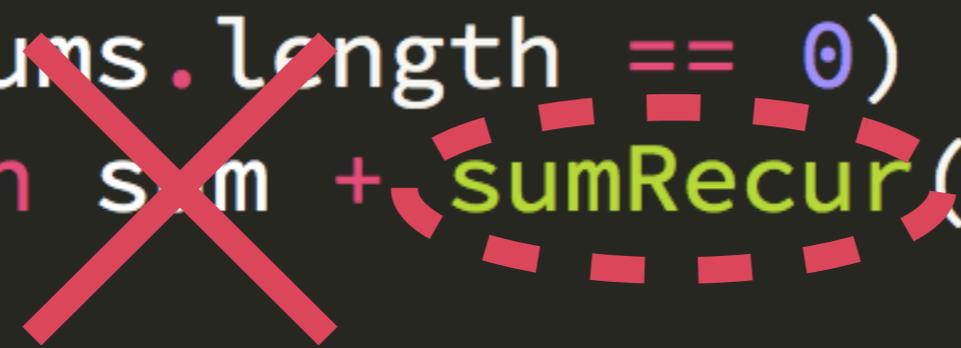
```
1 function sumIter(sum, ...nums) {  
2     for (var i=0; i<nums.length; i++) {  
3         sum = sum + nums[i];  
4     }  
5     return sum;  
6 }  
7  
8 sumIter(3,4,5,6,7,8,9); // 42
```

```
1 function sumRecur(sum, ...nums) {  
2     if (nums.length == 0) return sum;  
3     return sum + sumRecur(...nums);  
4 }  
5  
6 sumRecur(3,4,5,6,7,8,9); // 42
```

```
1 function sumRecur(sum, num, ...nums) {  
2     if (nums.length == 0) return sum + num;  
3     return sum + sumRecur(num, ...nums);  
4 }  
5  
6 sumRecur(3,4,5,6,7,8,9); // 42
```

EXERCISE 6

```
1 function sumRecur(sum, num, ...nums) {  
2     if (nums.length == 0) return sum + num;  
3     return sum + sumRecur(num, ...nums);  
4 }  
5  
6 sumRecur(3,4,5,6,7,8,9); // 42
```



PTC

PROPER TAIL CALLS

```
1 "use strict";
2
3 function foo(x) {
4     if (x < 10) return x;
5     return bar(x);
6 }
7
8 function bar(x) {
9     return x / 2;
10 }
11
12 foo(42); // 21
```

```
1 "use strict";
2
3 function foo(x) {
4     if (x % 2 == 1) {
5         x = Math.round(x / 3);
6     }
7     else {
8         x = x / 2;
9     }
10
11     if (x < 10) return x;
12     return foo(x);
13 }
14
15 foo(42); // 7
```

```
1 function sumRecur(sum, num, ...nums) {  
2     if (nums.length == 0) return sum + num;  
3     return sum + sumRecur(num, ...nums);  
4 }  
5  
6 sumRecur(3,4,5,6,7,8,9); // 42
```

sum + sum + sum + sum + ..

```
1 "use strict";
2
3 function sumRecur(...nums) {
4     return recur(...nums);
5
6     // *****
7     function recur(sum, num, ...nums) {
8         sum += num;
9         if (nums.length == 0) return sum;
10        return recur(sum, ...nums);
11    }
12 }
13
14 sumRecur(3,4,5,6,7,8,9); // 42
```

```
1 "use strict";
2
3 var sumRecur = (function(){
4     return function(...nums){
5         return recur(...nums);
6     };
7
8     // *****
9     function recur(sum, num, ...nums){
10        sum += num;
11        if (nums.length == 0) return sum;
12        return recur(sum, ...nums);
13    }
14 })();
15
16 sumRecur(3,4,5,6,7,8,9); // 42
```

```
1 "use strict";
2
3 function sumRecur(sum, num, ...nums) {
4     sum += num;
5     if (nums.length == 0) return sum;
6     return sumRecur(sum, ...nums);
7 }
8
9 sumRecur(3, 4, 5, 6, 7, 8, 9); // 42
```

CPS

```
1 "use strict";
2
3 var sumRecur = (function(...nums) {
4     return function(...nums) {
5         return recur(nums, v=>v);
6     };
7
8     // *****
9     function recur([sum, ...nums], cont) {
10        if (nums.length == 0) return cont(sum);
11        return recur(nums, function(v) {
12            return cont(sum + v);
13        });
14    }
15 })();
16
17 sumRecur(3,4,5,6,7,8,9); // 42
```

TRAMPOLINES

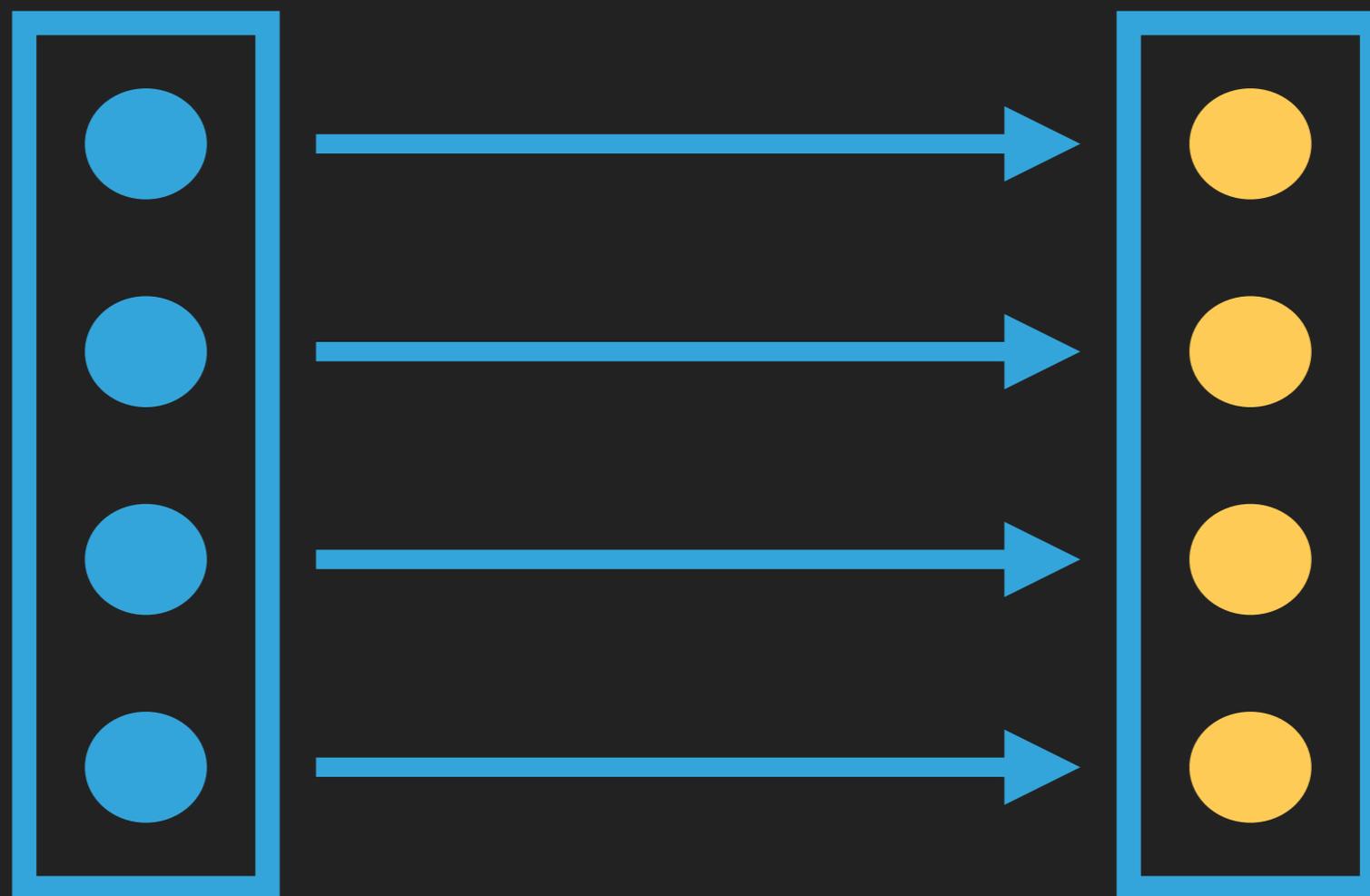
```
1 var sumTrampolined =
2   trampoline(function f(sum, num, ...nums) {
3     sum += num;
4     if (nums.length == 0) return sum;
5     return function() {
6       return f(sum, ...nums);
7     };
8   });
9
10 sumTrampolined(3,4,5,6,7,8,9); // 42
```

```
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 }
```


If you can do something awesome,
keep doing it repeatedly.

LISTS

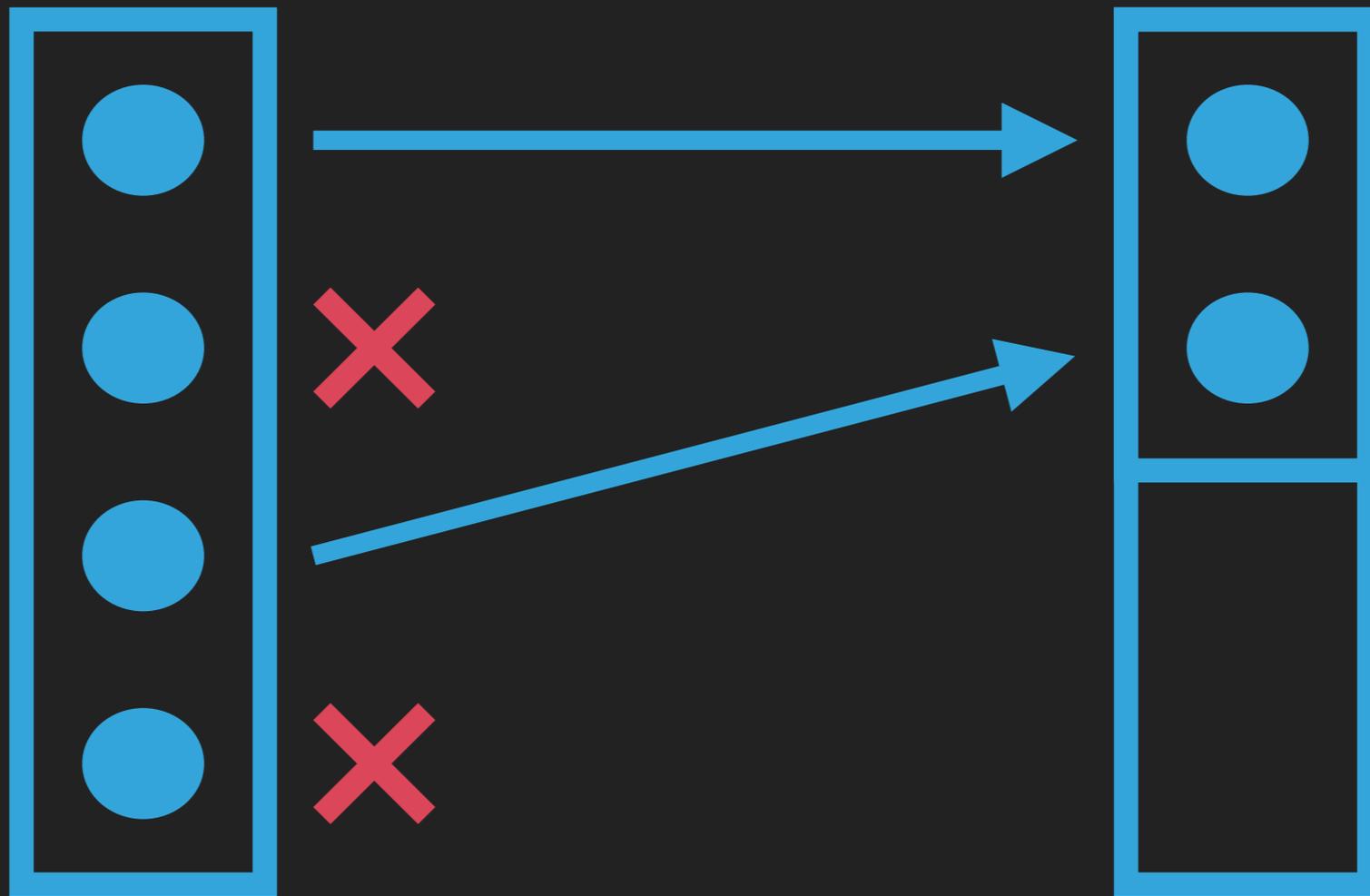
actually, data structures



MAP: TRANSFORMATION

```
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]
```

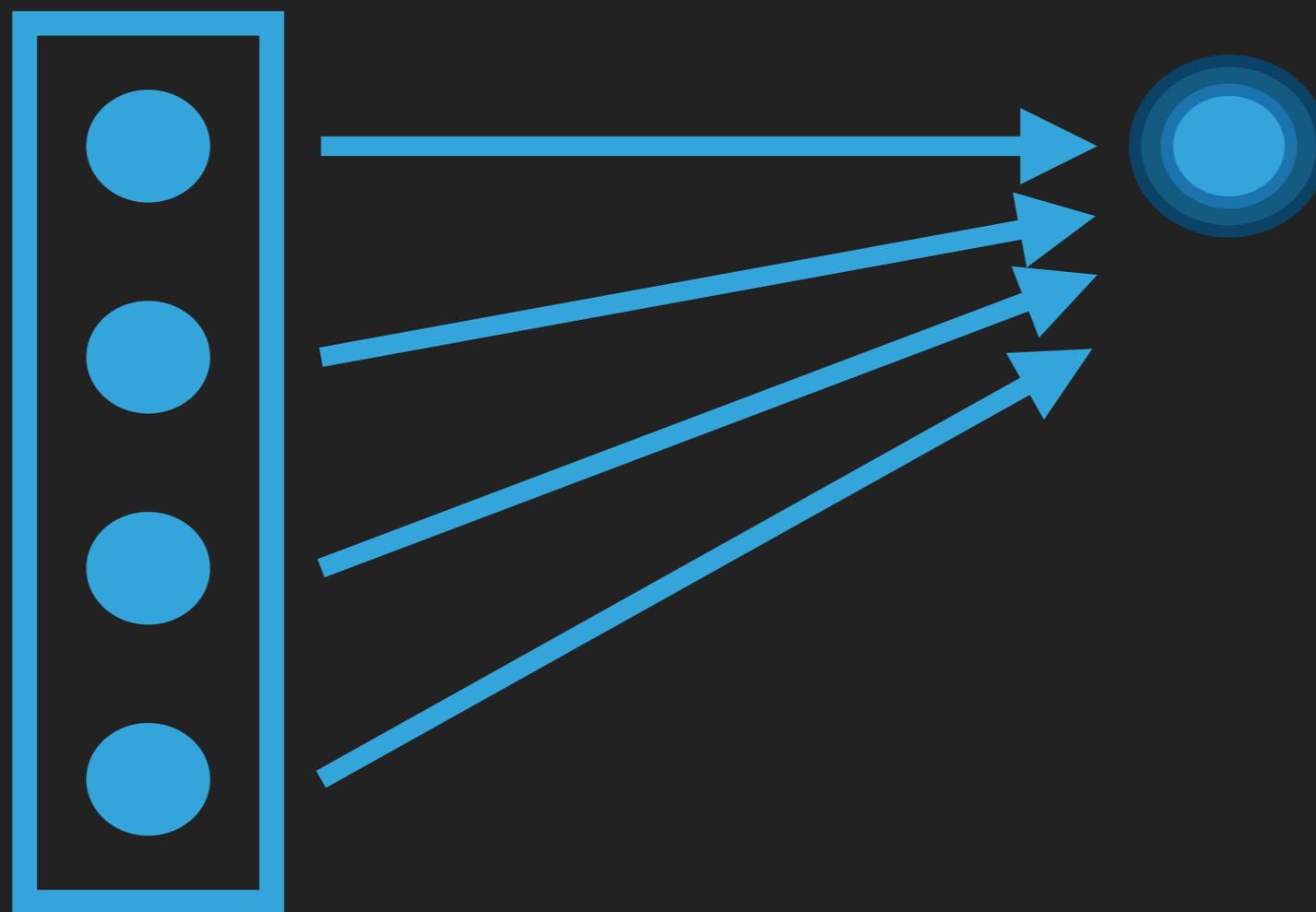
```
1 function doubleIt(val) {  
2     return val * 2;  
3 }  
4  
5 [1,2,3,4,5].map(doubleIt);  
6 // [2,4,6,8,10]
```



FILTER: EXCLUSION

```
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]
```

```
1 function onlyOdds(val) {  
2     return val % 2 == 1;  
3 }  
4  
5 [1,2,3,4,5].filter(onlyOdds);  
6 // [1,3,5]
```



REDUCE: COMBINING

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

```
1 function acronym(str, word) {
2     return str + word.charAt(0);
3 }
4
5 ["Functional", "Light", "JavaScript", "Stuff"]
6 .reduce(acronym, "");
7 // FLJS
```

EXERCISE 7

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 function composeRight(fn1, fn2) {
6     return function(...args) {
7         return fn1(fn2(...args));
8     };
9 }
10
11 var list = [2, 5, 8, 11, 14, 17, 20];
12
13 list
14 .map(
15     [div3, mul2, add1].reduce(composeRight)
16 );
17 // [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 = [2, 5, 8, 11, 14, 17, 20];
6
7 list
8   .map( add1 )
9   .filter( isOdd )
10  .reduce( sum );
11 // 48
```

```
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 = [2,5,8,11,14,17,20];
16
17 list = mapWithReduce( list, add1 );
18 list = filterWithReduce( list, isOdd );
19 list.reduce( sum );
20 // 48
```

```
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 = [2,5,8,11,14,17,20];
16
17 list
18 .reduce( mapReducer(add1), [] )
19 .reduce( filterReducer(isOdd), [] )
20 .reduce( sum );
21 // 48
```

```
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 = [2,5,8,11,14,17,20];
20
21 list
22 .reduce( mapReducer(add1), [] )
23 .reduce( filterReducer(isOdd), [] )
24 .reduce( sum );
25 // 48
```

```
1 function listCombination(list,v) {
2     list.push(v);
3     return list;
4 }
5
6 var mapReducer = curry(function mapReducer(mappingFn, combineFn) {
7     return function reducer(list,v) {
8         return combineFn( list, mappingFn(v) );
9     };
10 });
11
12 var filterReducer = curry(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 = [2,5,8,11,14,17,20];
20
21 list
22 .reduce( mapReducer(add1, listCombination), [] )
23 .reduce( filterReducer(isOdd, listCombination), [] )
24 .reduce( sum );
25 // 48
```

```
1 function listCombination(list,v) {
2     list.push(v);
3     return list;
4 }
5
6 var mapReducer = curry(function mapReducer(mappingFn, combineFn) {
7     return function reducer(list,v) {
8         return combineFn( list, mappingFn(v) );
9     };
10 });
11
12 var filterReducer = curry(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 = [2,5,8,11,14,17,20];
22
23 list
24 .reduce( transducer(listCombination), [] )
25 .reduce( sum );
26 // 48
```

```
1  var mapReducer = curry(function mapReducer(mappingFn, combineFn) {
2      return function reducer(list, v) {
3          return combineFn( list, mappingFn(v) );
4      };
5  });
6
7  var filterReducer = curry(function filterReducer(predicateFn, combineFn) {
8      return function reducer(list, v) {
9          if (predicateFn(v)) return combineFn( list, v );
10         return list;
11     };
12 });
13
14 var transducer = compose( mapReducer(add1), filterReducer(isOdd) );
15
16 var list = [2,5,8,11,14,17,20];
17
18 list
19 .reduce( transducer(sum), 0 );
20 // 48
```

```
1 function transduce(transducer, combineFn, initialValue, list) {
2     var reducer = transducer(combineFn);
3     return list.reduce(reducer, initialValue);
4 }
5
6 var transducer = compose( mapReducer(add1), filterReducer(isOdd) );
7
8 transduce( transducer, sum, 0, [2,5,8,11,14,17,20] );
9 // 48
```

DATA STRUCTURE OPERATIONS

```
1  function mapObj(mapperFn, o) {
2      var newObj = {};
3      var keys = Object.keys(o);
4      for (let key of keys) {
5          newObj[key] = mapperFn( o[key] );
6      }
7      return newObj;
8  }
9
10 var obj = {
11     a: "Hello",
12     b: "World"
13 };
14
15 mapObj(function upper(val) {
16     return val.toUpperCase();
17 }, o);
18 // {a: "HELLO", b: "WORLD" }
```

EXERCISE 8

 [getify / fpo](#)

 Code

 Issues **2**

 Pull requests **1**

 Projects **0**

 V

FP library for JavaScript. Supports named-argument style methods.

[library](#)

[functional-programming](#)

[functional-js](#)

[javascript](#)

[Manage topics](#)

github.com/getify/fpo

```
1 // the classic/traditional method style
2 // (on the `FP0.std.*` namespace)
3 FP0.std.reduce(
4     (acc, v) => acc + v,
5     undefined,
6     [3, 7, 9]
7 ); // 19
8
9 // FP0 named-argument method style
10 FP0.reduce({
11     arr: [3, 7, 9],
12     fn: ({acc, v}) => acc + v
13 }); // 19
```

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

```
1 var f = FP0.reduce({ arr: [3,7,9] });  
2  
3 // later:  
4 f({ fn: ({acc, v}) => acc + v }); // 19  
5 f({ fn: ({acc, v}) => acc * v }); // 189
```


ASYN

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

FP OVER TIME

EAGER

VS

LAZY

```
1 var a = [];  
2  
3 var b = mapLazy(a, function(v) {  
4     return v * 2;  
5 });  
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(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(v) {
10     return v * 2;
11 });
12
13 b.subscribe(function onValue(v) {
14     console.log(v);
15 });
```

EXERCISE 9

RECAP:

- ▶ Functions (~~side effects~~, point-free)
- ▶ Composition
- ▶ Immutability
- ▶ Closure
- ▶ List & Data Structure operations
- ▶ Async (observables)

THANKS!!!!

KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS