

KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS

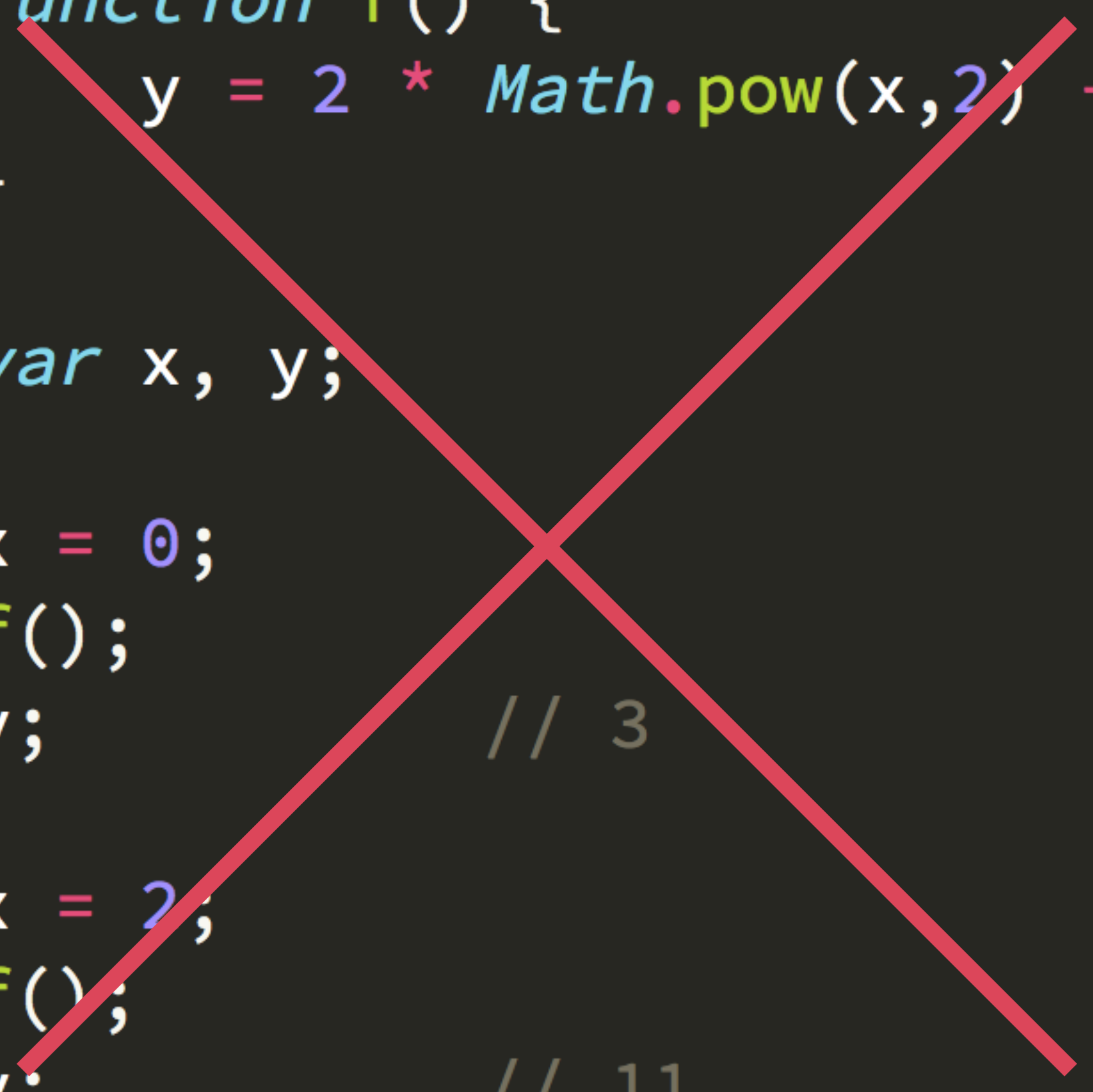
FUNCTIONS

```
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
```

```
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 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  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

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 x_y = mult( 3, 4 );  
11 sum( x_y, 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 // 5 + (3 * 4)  
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 // 5 + (3 * 4)
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 compose2(fn1,fn2) {
10     return function comp(arg1,arg2,arg3){
11         return fn2(
12             fn1( arg1, arg2 ),
13             arg3
14         );
15     };
16 }
17
18 var multAndSum = compose2(mult,sum);
19
20 // 5 + (3 * 4)
21 multAndSum(3,4,5);           // 17
```

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

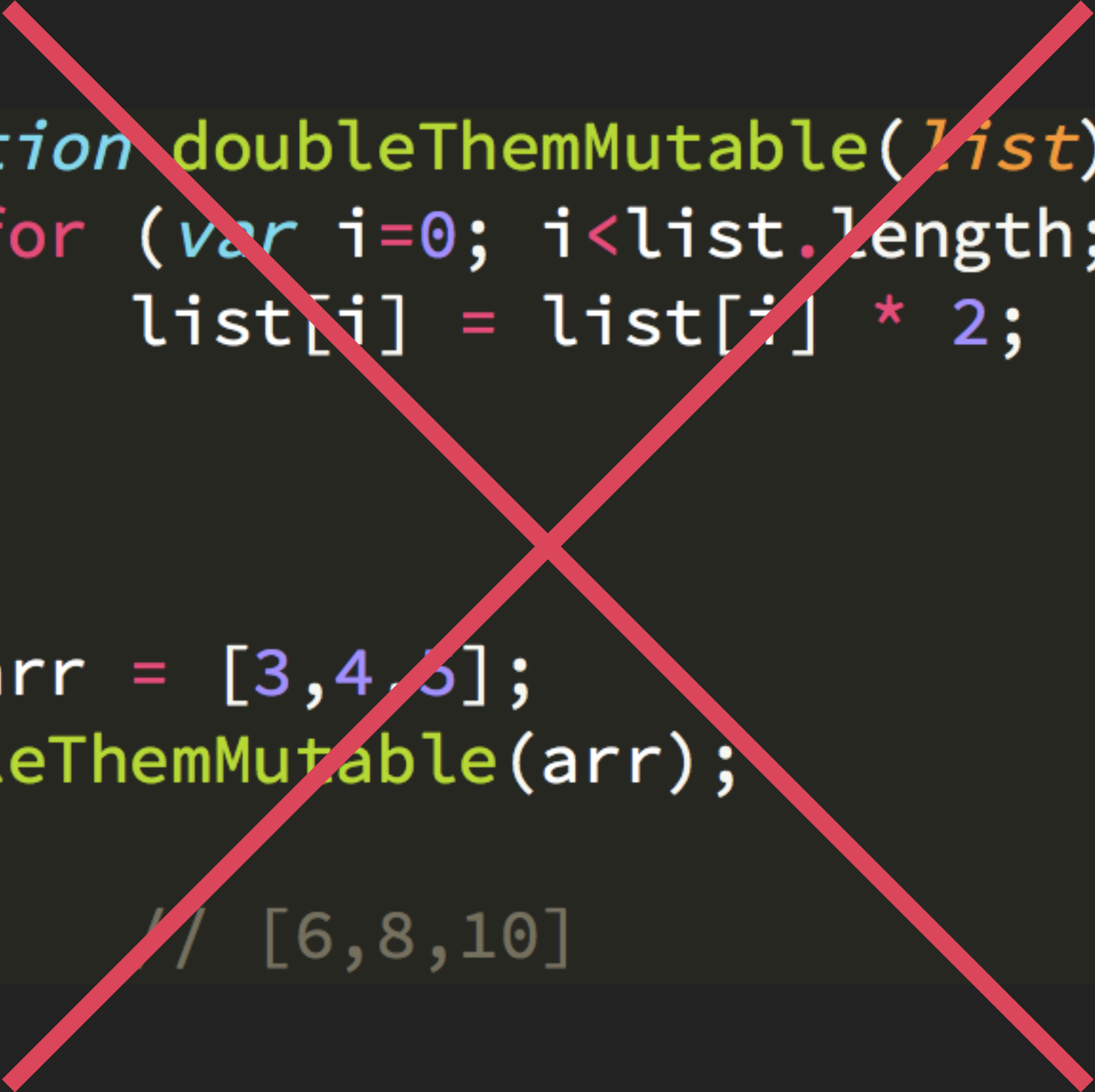
```
1  function increment(x) {
2      return x + 1;
3  }
4
5  function double(x) {
6      return x * 2;
7  }
8
9  var f = composeRight(increment, double);
10 var p = composeRight(double, increment);
11
12 f(3);    // 7
13 p(3);    // 8
```

EXERCISE 2

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



```
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]
```

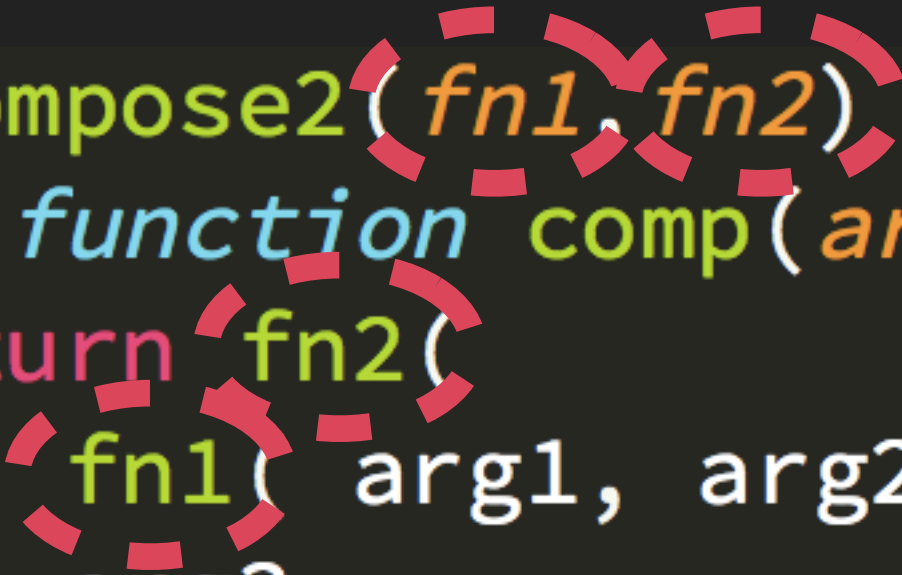


```
1 function doubleThemImmutable(list) {  
2     var newList = [];  
3     for (var i=0; i<list.length; i++) {  
4         newList[i] = list[i] * 2;  
5     }  
6     return newList;  
7 }  
8  
9 var arr = [3,4,5];  
10 var arr2 = doubleThemImmutable(arr);  
11  
12 arr;      // [3,4,5]  
13 arr2;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

CLOSURE

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

```
1  function compose2(fn1, fn2) {  
2      return function comp(arg1, arg2, arg3) {  
3          return fn2(  
4              fn1( arg1, arg2 ),  
5                  arg3  
6          );  
7      };  
8  }
```

A diagram illustrating function composition. Two red dashed circles are drawn over the code. The first circle is centered on the arguments *fn1* and *fn2* in the function signature of `compose2` on line 1. The second circle is centered on the arguments *fn1* and *arg3* in the nested function `comp` on line 3. This visualizes how `fn1` is passed to `comp` and how `fn2` is the function that `comp` returns.

```
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 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 }
```

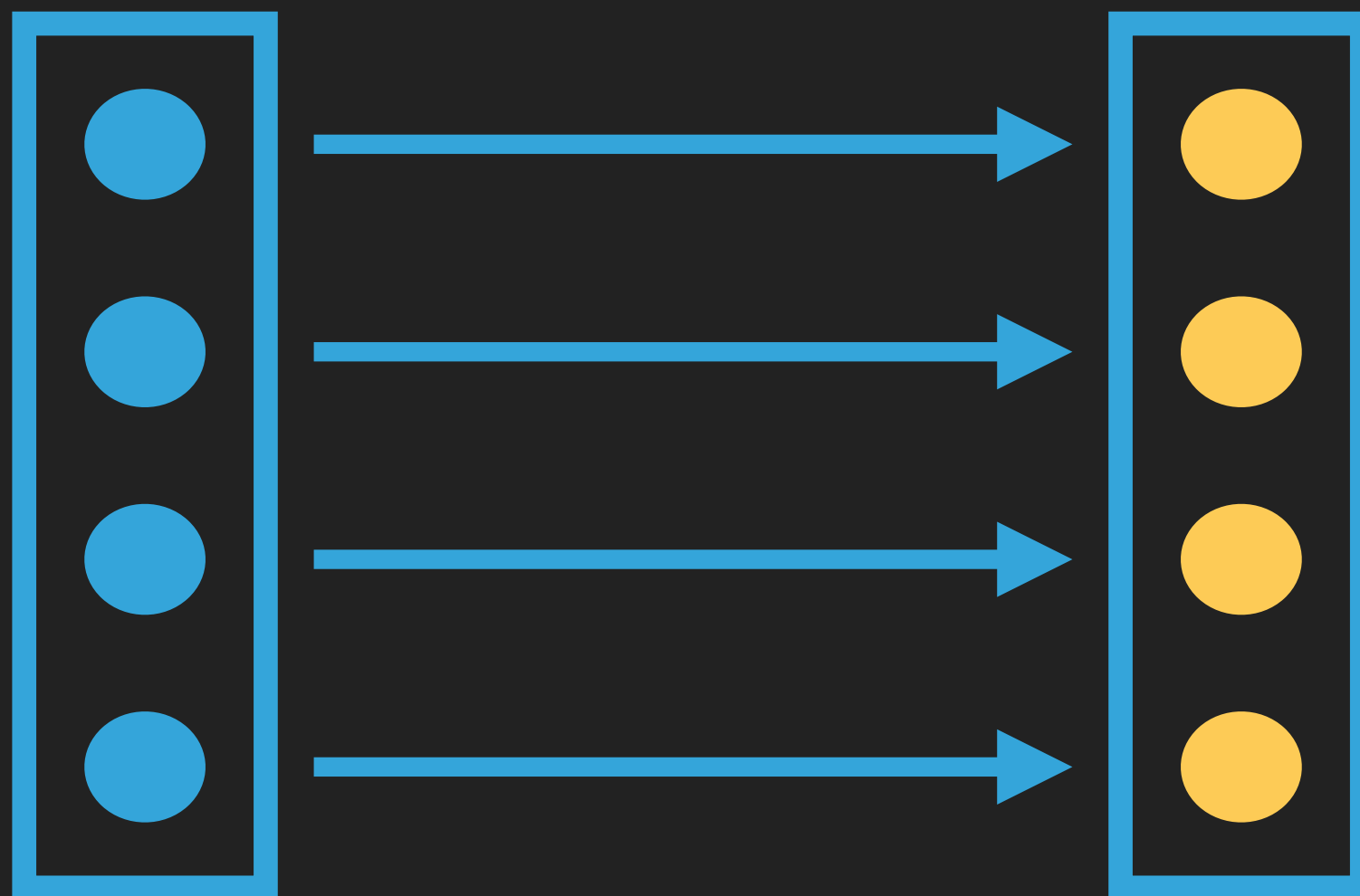
```
10  
11 var addTo10 = partial(add,10);  
12  
13 addTo10(32);    // 42
```

```
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
```

EXERCISE 3

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

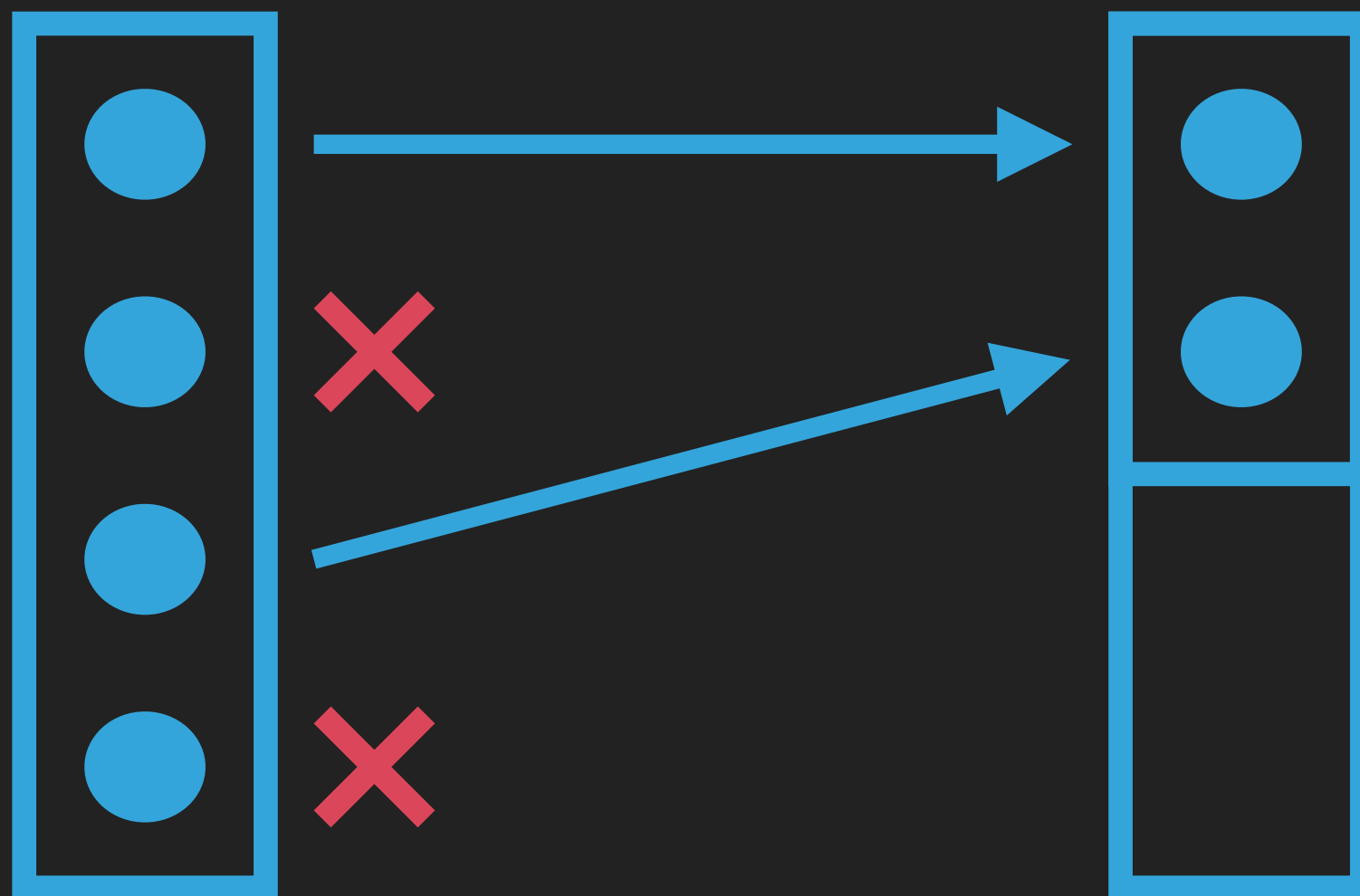
LISTS



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]
```

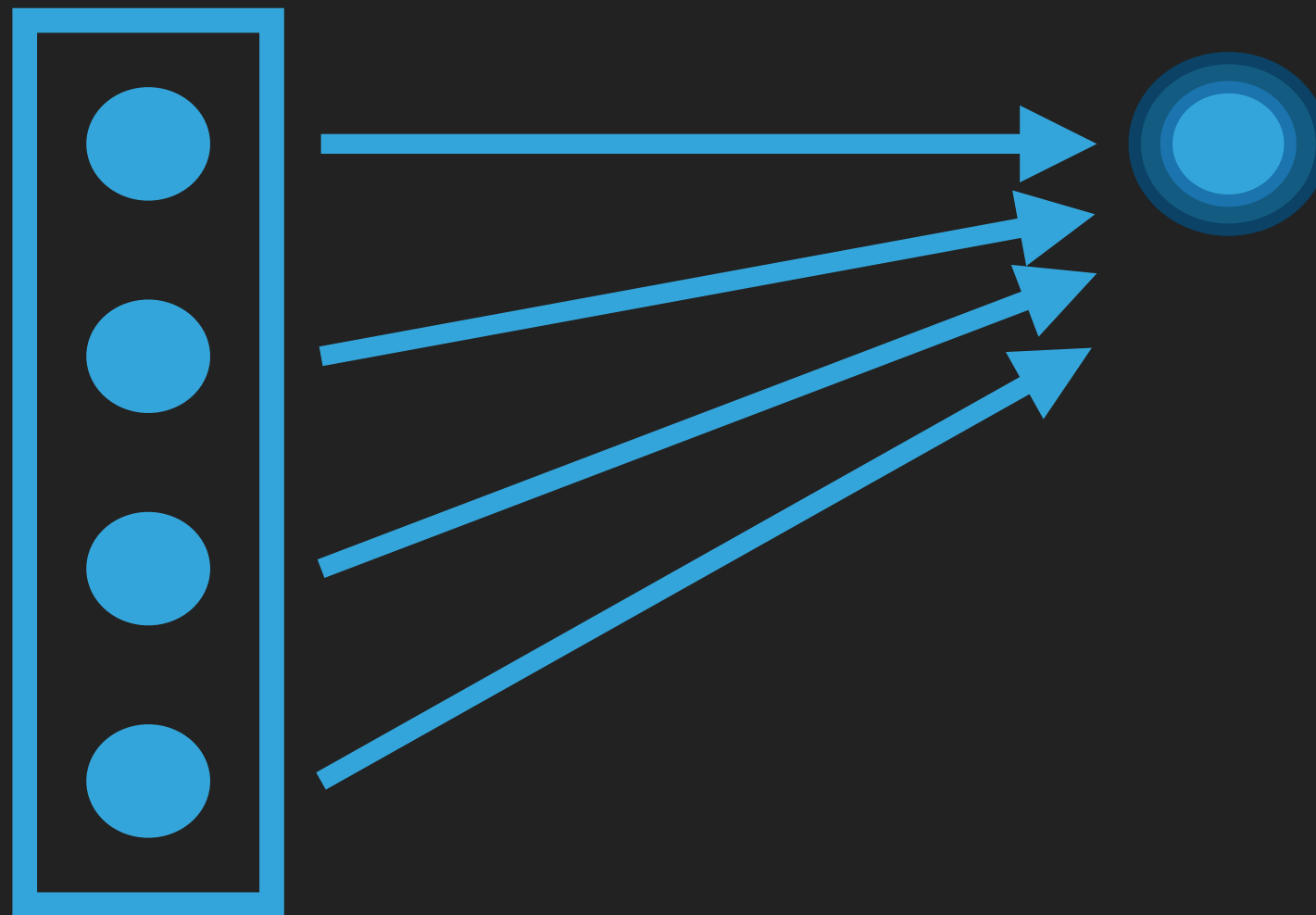
```
1 function doubleIt(val) {  
2     return val * 2;  
3 }  
4  
5 [1,2,3,4,5].map(doubleIt);  
6 // [2,4,6,8,10]
```



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 4

EXTRA CREDIT

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]
```

TRANSDUCE


```
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 filterWithReduce(
18     mapWithReduce( list, add1 ),
19     isOdd
20 )
21 .reduce( sum );
22 // 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  function mapReducer(mappingFn) {
7      return function toCombine(combineFn){
8          return function reducer(list,v){
9              return combineFn( list, mappingFn(v) );
10         };
11     };
12 }
13
14 function filterReducer(predicateFn) {
15     return function toCombine(combineFn){
16         return function reducer(list,v){
17             if (predicateFn(v)) return combineFn( list, v );
18             return list;
19         };
20     };
21 }
22
23 var list = [2,5,8,11,14,17,20];
24
25 list
26 .reduce( mapReducer(add1)(listCombination), [] )
27 .reduce( filterReducer(isodd)(listCombination), [] )
28 .reduce( sum );
29 // 48
```



```
1  function listCombination(list,v) {
2      list.push(v);
3      return list;
4  }
5
6  function mapReducer(mappingFn) {
7      return function toCombine(combineFn){
8          return function reducer(list,v){
9              return combineFn( list, mappingFn(v) );
10         };
11     };
12 }
13
14 function filterReducer(predicateFn) {
15     return function toCombine(combineFn){
16         return function reducer(list,v){
17             if (predicateFn(v)) return combineFn( list, v );
18             return list;
19         };
20     };
21 }
22
23 var transducer =
24     composeRight( mapReducer(add1), filterReducer(isOdd) )(listCombination);
25
26 var list = [2,5,8,11,14,17,20];
27
28 list
29 .reduce( transducer, [] )
30 .reduce( sum );
31 // 48
```

```
1  function mapReducer(mappingFn) {
2      return function toCombine(combineFn){
3          return function reducer(list,v){
4              return combineFn( list, mappingFn(v) );
5          };
6      };
7  }
8
9  function filterReducer(predicateFn) {
10     return function toCombine(combineFn){
11         return function reducer(list,v){
12             if (predicateFn(v)) return combineFn( list, v );
13             return list;
14         };
15     };
16 }
17
18 var transducer =
19     composeRight( mapReducer(add1), filterReducer(isOdd) )(sum);
20
21 var list = [2,5,8,11,14,17,20];
22
23 list
24 .reduce( transducer, 0)
25 // 48
```

RECAP:

- ▶ Pure Functions (~~side effects~~)
- ▶ Composition
- ▶ Immutability
- ▶ Closure
- ▶ Lists (map, filter, reduce)
(fusion, transducing)

THANKS!!!!

KYLE SIMPSON GETIFY@GMAIL.COM

FUNCTIONAL-LIGHT JS