

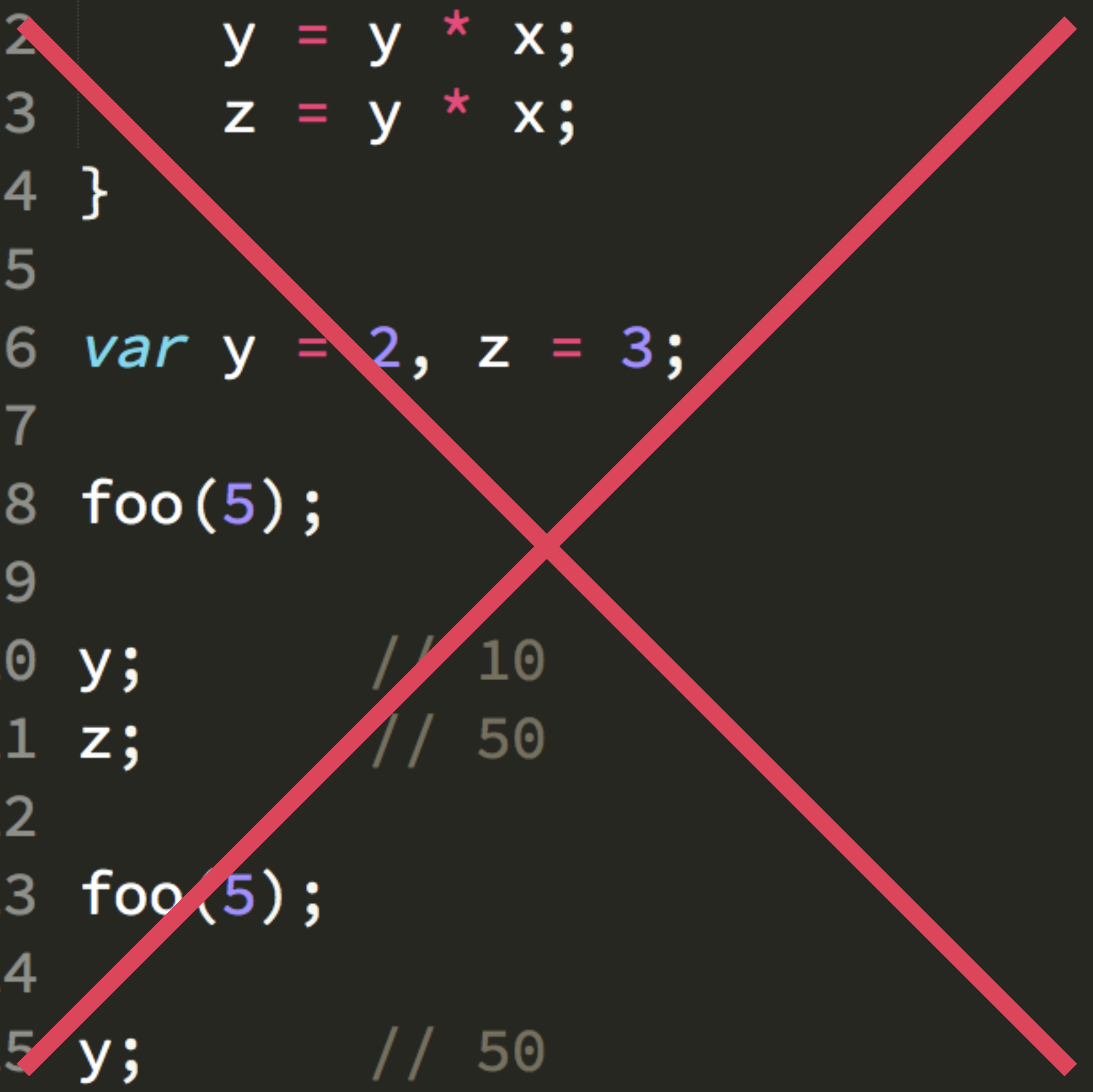
KYLE SIMPSON @GETIFY

---

# FUNCTIONAL-LIGHT JAVASCRIPT

~~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;           // 50  
12  
13 foo(5);  
14  
15 y;           // 50  
16 z;           // 250
```



**PURE FUNCTIONS**

```
1  function bar(x,y,z) {
2      foo(x);
3      return [y,z];
4
5      // *****
6
7      function foo(x) {
8          y = y * x;
9          z = y * x;
10     }
11 }
12
13 bar(5,2,3);           // [10,50]
14 bar(5,10,50);         // [50,250]
```

**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
```



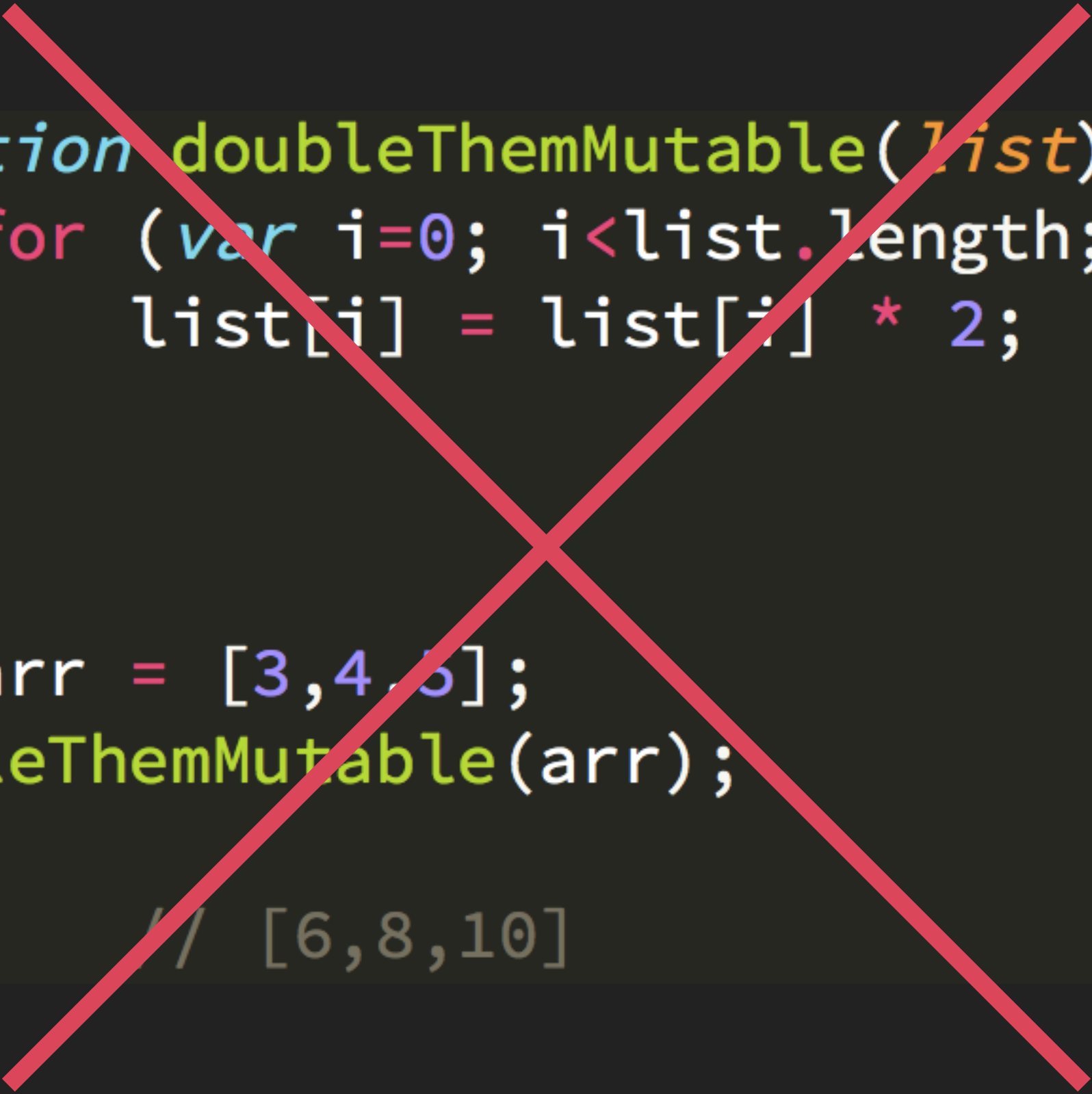
```
1  function sum(x,y) {
2      return x + y;
3  }
4
5  function mult(x,y) {
6      return x * y;
7  }
8
9  function multAndSum(x,y,z) {
10     return sum( mult( x, y ), z );
11 }
12
13 // 5 + (3 * 4)
14 multAndSum(3,4,5);           // 17
```

```
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
```

**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]
```

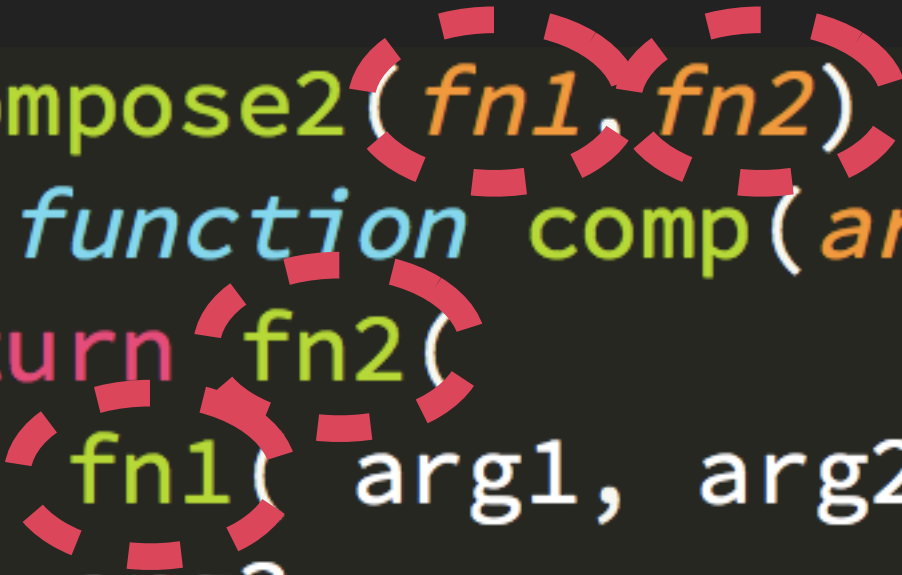
```
1 function doubleThemImmutable(list) {  
2     var newList = [];  
3     for (var i=0; i<list.length; i++) {  
4         newList[i] = list[i] * 2;  
5     }  
6     return newList;  
7 }  
8  
9 var arr = [3,4,5];  
10 var arr2 = doubleThemImmutable(arr);  
11  
12 arr;      // [3,4,5]  
13 arr2;     // [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 call to *fn1* on line 4, specifically around the arguments `arg1` and `arg2`. This visualizes how the output of *fn1* is passed as an argument to *fn2*.

```
1  function add(x,y) {
2      return x + y;
3  }
4
5  function curry(fn,...args) {
6      return function(lastArg) {
7          return fn(...args,lastArg);
8      };
9  }
10
11  var addTo10 = curry(add,10);
12
13  addTo10(32);           // 42
```

# RECURSION

```
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
```



# PTC

## PROPER TAIL CALLS

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

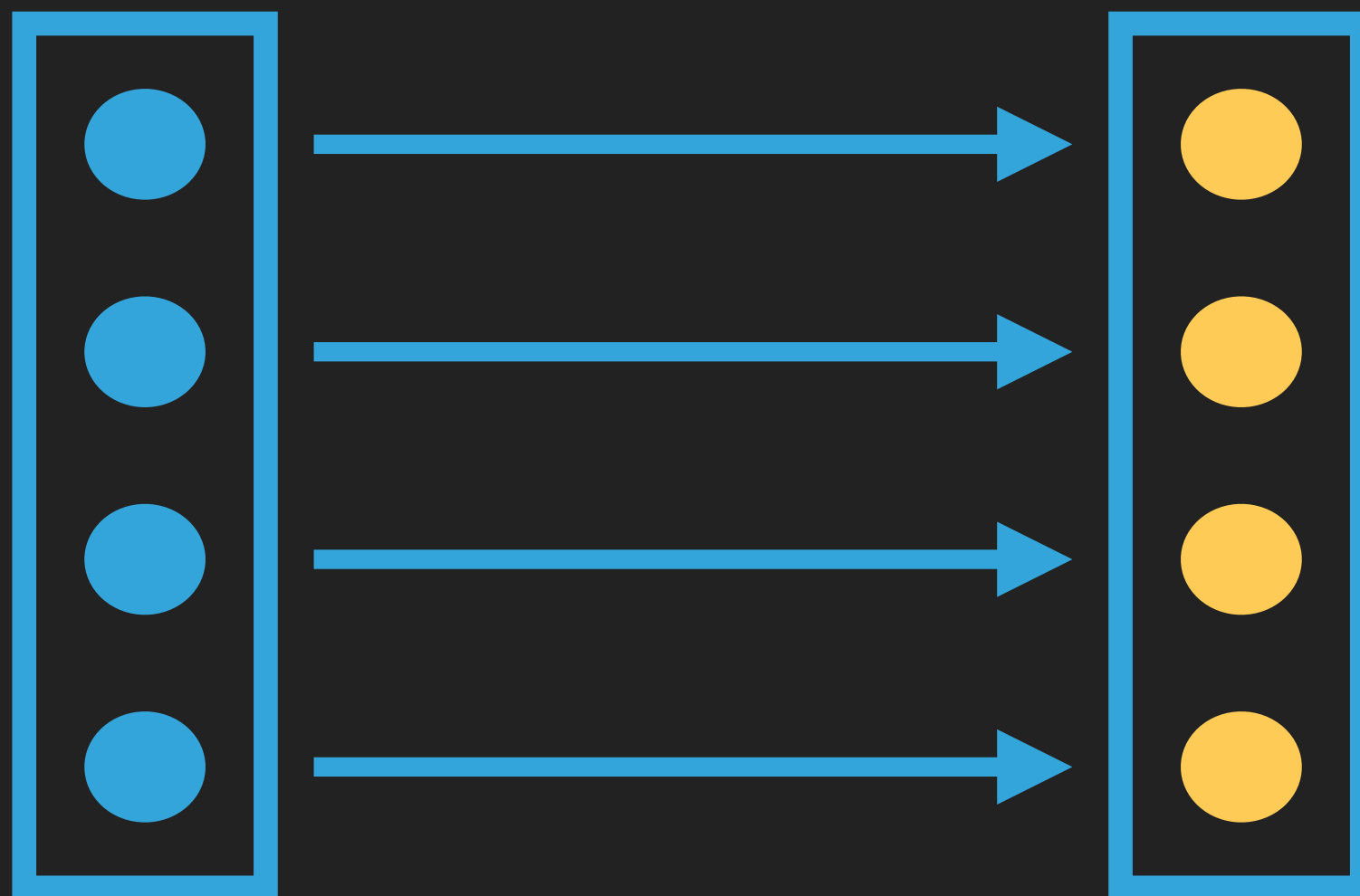


```
1  const sumRecur = (function(){
2      return (...nums) => recur(...nums);
3
4      // *****
5      function recur(sum, num, ...nums) {
6          sum += num;
7          if (nums.length == 0) return sum;
8          return recur(sum, ...nums);
9      }
10 }());
11
12 sumRecur(3,4,5,6,7,8,9);    //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

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

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

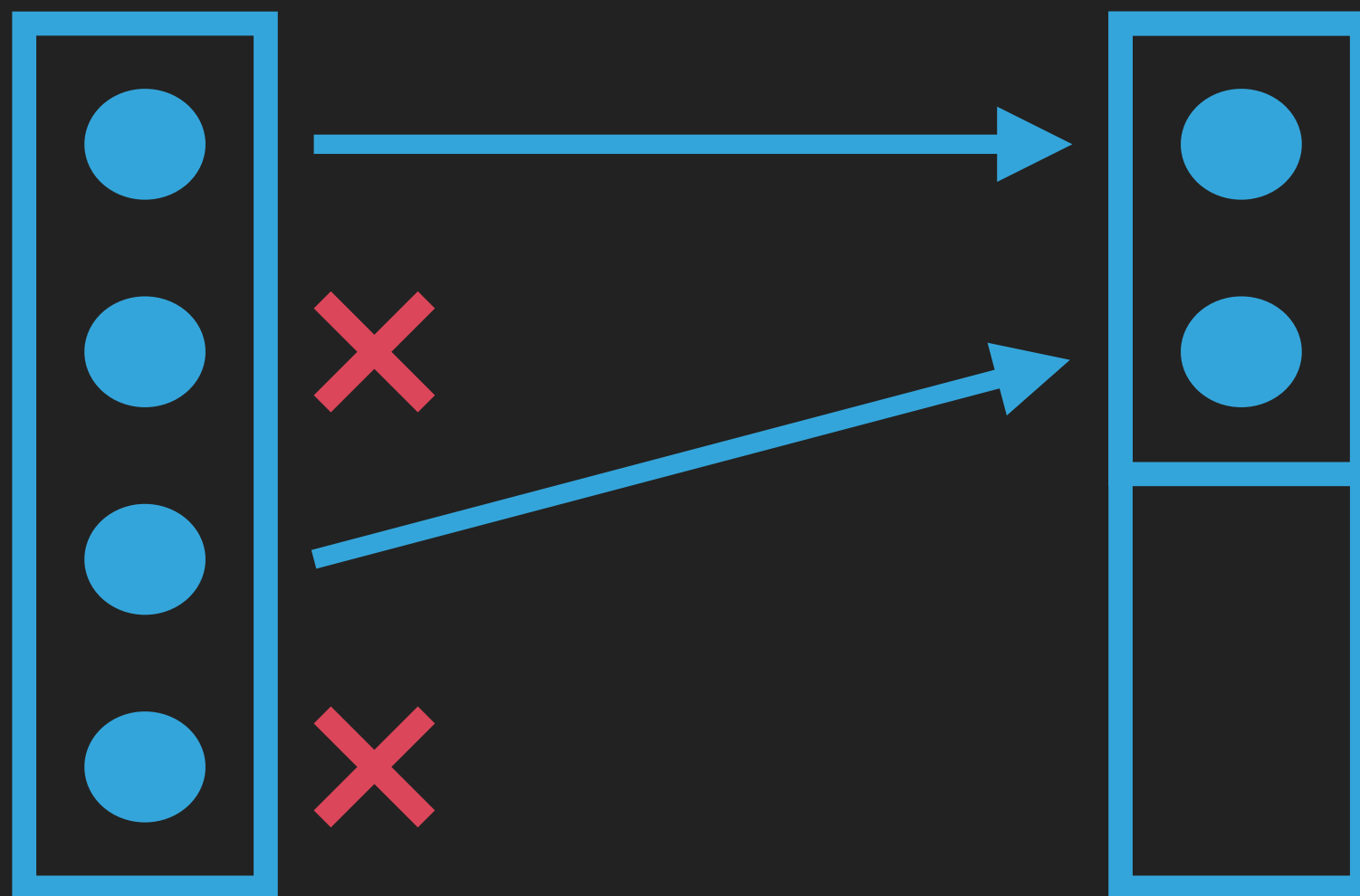
# LISTS



**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]
```

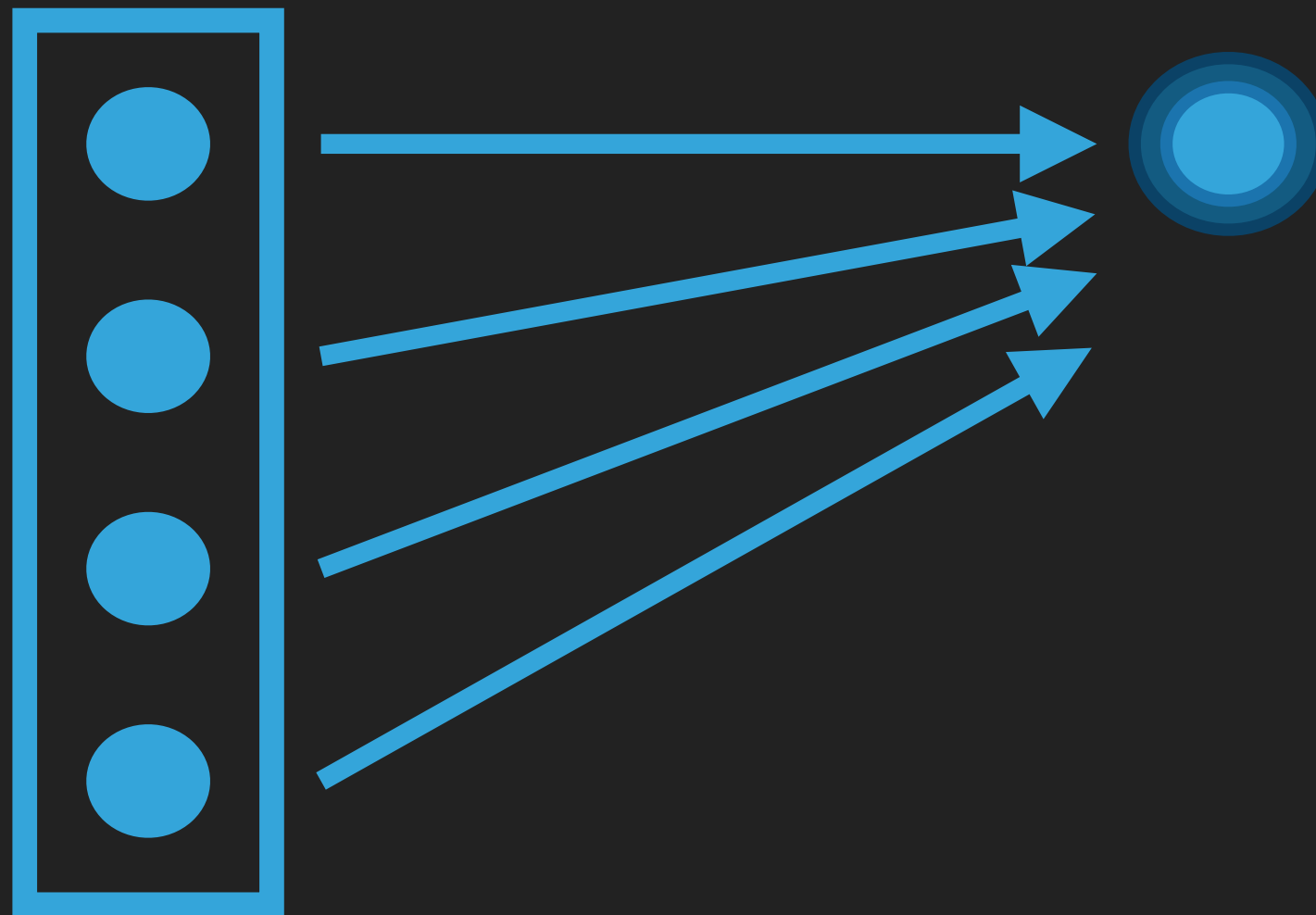


**FILTER: EXCLUSION**

```
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
```

**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]
```

A diagram consisting of three dashed red circles arranged vertically, each enclosing a '.map' call in the code. The first circle is around line 8, the second around line 9, and the third around line 10. This visualizes the sequential application of the map function to the list.

```
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
- ▶ Recursion
- ▶ Lists (map, filter, reduce)  
(fusion, transducing)



KYLE SIMPSON @GETIFY

---

# FUNCTIONAL-LIGHT JAVASCRIPT