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

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

function mult(x,y) {
    return x * y;
}

// 5 + (3 * 4)
var x_y = mult( 3, 4 );
sum( x_y, 5 );                    // 17
```

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

function mult(x,y) {
    return x * y;
}

// 5 + (3 * 4)
sum( mult( 3, 4), 5 );  // 17
```

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

function mult(x,y) {
    return x * y;
}

function multAndSum(x,y,z) {
    return sum( mult( x, y ), z );
}

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

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

function mult(x,y) {
    return x * y;
}

function compose2(fn1,fn2) {
    return function comp(arg1,arg2,arg3){
        return fn2(
            fn1( arg1, arg2 ),
            arg3
        );
    };
}

var multAndSum = compose2(mult,sum);

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

IMMUTABILITY

```
var x = 2;
x++;                        // allowed

const y = 3;
y++;                        // not allowed

const z = [4,5,6];
z = 10;                     // not allowed
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]
```

```
function doubleThemImmutable(list) {
    var newList = [];
    for (var i=0; i<list.length; i++) {
        newList[i] = list[i] * 2;
    }
    return newList;
}

var arr = [3,4,5];
var arr2 = doubleThemImmutable(arr);

arr;     // [3,4,5]
arr2;    // [6,8,10]
```

CLOSURE

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

```
function compose2(fn1,fn2) {
    return function comp(arg1,arg2,arg3){
        return fn2(
            fn1( arg1, arg2 ),
            arg3
        );
    };
}
```

```
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

```
function sumIter(sum,...nums) {
    for (var i=0; i<nums.length; i++) {
        sum = sum + nums[i];
    }
    return sum;
}

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

```
function sumRecur(...nums) {
    return recur(...nums);

    // **************************
    function recur(sum,num,...nums) {
        sum += num;
        if (nums.length == 0) return sum;
        return recur(sum,...nums);
    }
}

sumRecur(3,4,5,6,7,8,9);     // 42
```

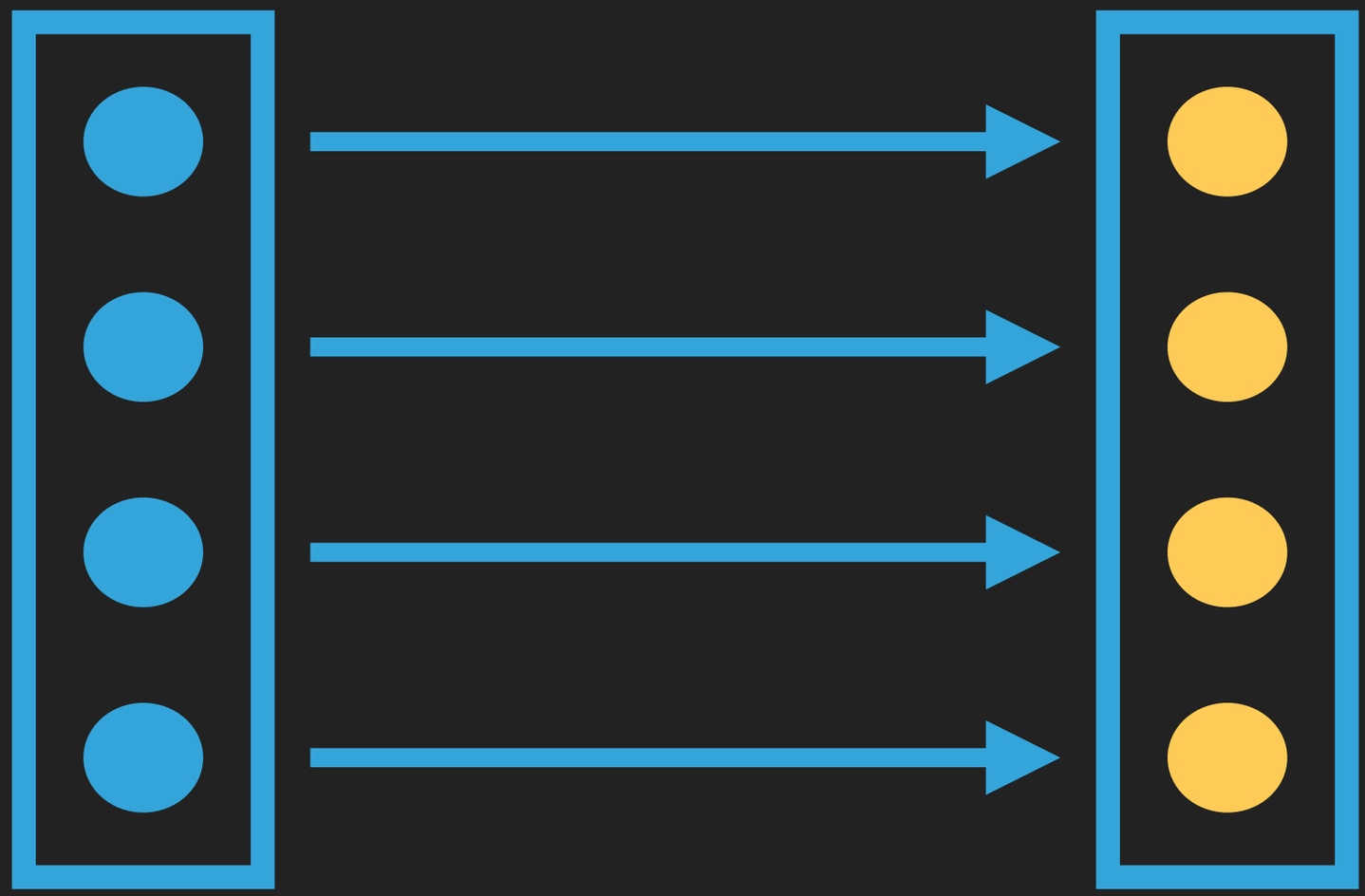
```
1  const sumRecur = (function(){
2      return (...nums) => recur(...nums);
3
4      // **************************
5      function recur(sum,num,...nums) {
6          sum += num;
7          if (nums.length == 0) return sum;
8          return recur(sum,...nums);
9      }
10 }();
11
12 sumRecur(3,4,5,6,7,8,9);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

```
function sumRecur(sum,num,...nums) {
    sum += num;
    if (nums.length == 0) return sum;
    return sumRecur(sum,...nums);
}

sumRecur(3,4,5,6,7,8,9);     // 42
```

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

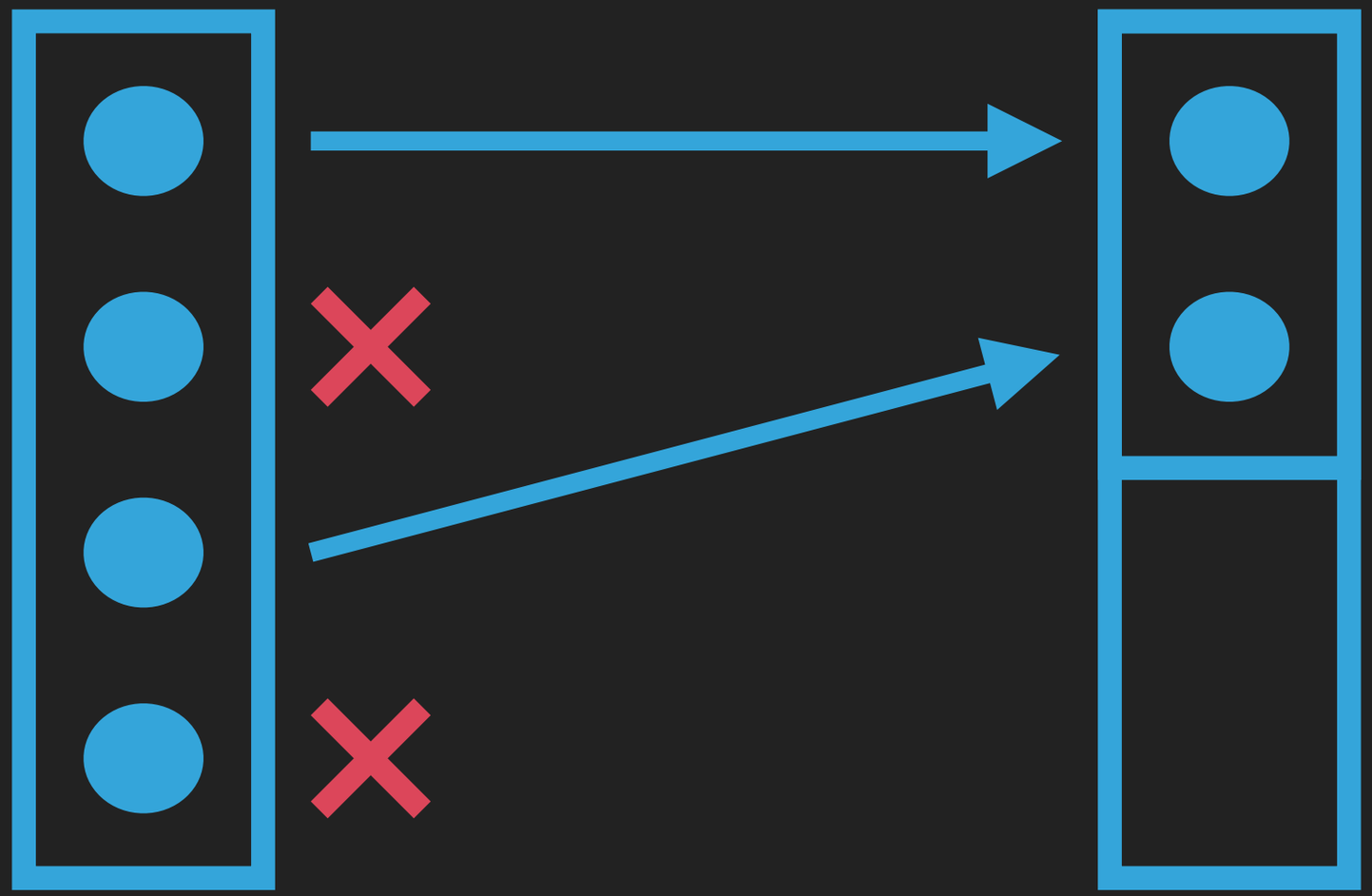
LISTS



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

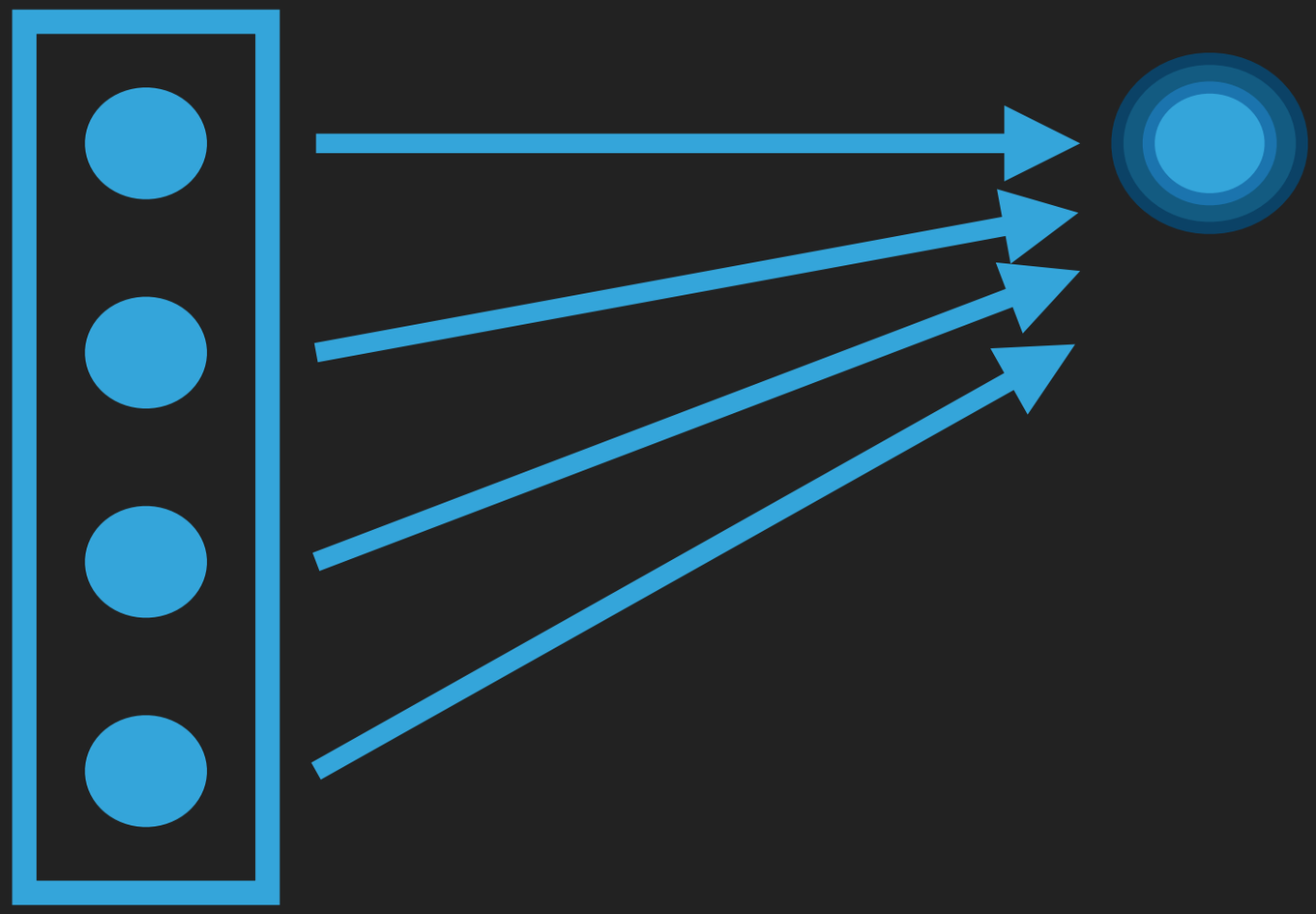
```
function isOdd(v) { return v % 2 == 1; }

function exclude(arr,fn) {
    var list = [];
    for (var i=0; i<arr.length; i++) {
        if (fn(arr[i])) {
            list.push(arr[i]);
        }
    }
    return list;
}


exclude([1,2,3,4,5],isOdd);
// [1,3,5]
```

```
function onlyOdds(val) {
    return val % 2 == 1;
}

[1,2,3,4,5].filter(onlyOdds);
// [1,3,5]
```



REDUCE: COMBINING

```
function mult(x,y) { return x * y; }

function combine(arr,fn,initial) {
    var result = initial;
    for (var i=0; i<arr.length; i++) {
        result = fn(result,arr[i]);
    }
    return result;
}

combine([1,2,3,4,5],mult,1);
// 120
```

```
function acronym(str,word) {
    return str + word.charAt(0);
}

["Functional","Light","JavaScript","Stuff"]
.reduce(acronym,"");
// 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]
```

```
function add1(v) { return v + 1; }
function mul2(v) { return v * 2; }
function div3(v) { return v / 3; }

function composeRight(fn1,fn2) {
    return function(...args){
        return fn1(fn2(...args));
    };
}

var list = [2,5,8,11,14,17,20];

list
.map(
    [div3,mul2,add1].reduce(composeRight)
);
// [2,4,6,8,10,12,14]
```

TRANSDUCE

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

var list = [2,5,8,11,14,17,20];

list
.map( add1 )
.filter( isOdd )
.reduce( sum );
// 48
```

```
function mapWithReduce(arr,mappingFn) {
    return arr.reduce(function reducer(list,v){
        list.pusn( mappingFn(v) );
        return list;
    }, [] );
}

function filterWithReduce(arr,predicateFn) {
    return arr.reduce(function reducer(list,v){
        if (predicateFn(v)) list.push(v);
        return list;
    }, [] );
}

var list = [2,5,8,11,14,17,20];

filterWithReduce(
    mapWithReduce( list, add1 ),
    isOdd
)
.reduce( sum );
// 48
```

```
function mapReducer(mappingFn) {
    return function reducer(list,v){
        list.push( mappingFn(v) );
        return list;
    };
}

function filterReducer(predicateFn) {
    return function reducer(list,v){
        if (predicateFn(v)) list.push(v);
        return list;
    };
}

var list = [2,5,8,11,14,17,20];

list
.reduce( mapReducer(add1), [] )
.reduce( filterReducer(isOdd), [] )
.reduce( sum );
// 48
```

```
function listCombination(list,v) {
    list.push(v);
    return list;
}

function mapReducer(mappingFn) {
    return function reducer(list,v){
        return listCombination( list, mappingFn(v) );
    };
}

function filterReducer(predicateFn) {
    return function reducer(list,v){
        if (predicateFn(v)) return listCombination( list, v );
        return list;
    };
}

var list = [2,5,8,11,14,17,20];

list
.reduce( mapReducer(add1), [] )
.reduce( filterReducer(isOdd), [] )
.reduce( sum );
// 48
```

```
function listCombination(list,v) {
    list.push(v);
    return list;
}

function mapReducer(mappingFn) {
    return function toCombine(combineFn){
        return function reducer(list,v){
            return combineFn( list, mappingFn(v) );
        };
    };
}

function filterReducer(predicateFn) {
    return function toCombine(combineFn){
        return function reducer(list,v){
            if (predicateFn(v)) return combineFn( list, v );
            return list;
        };
    };
}

var list = [2,5,8,11,14,17,20];

list
.reduce( mapReducer(add1)(listCombination), [] )
.reduce( filterReducer(isOdd)(listCombination), [] )
.reduce( sum );
// 48
```

```
function listCombination(list,v) {
    list.push(v);
    return list;
}

function mapReducer(mappingFn) {
    return function toCombine(combineFn){
        return function reducer(list,v){
            return combineFn( list, mappingFn(v) );
        };
    };
}

function filterReducer(predicateFn) {
    return function toCombine(combineFn){
        return function reducer(list,v){
            if (predicateFn(v)) return combineFn( list, v );
            return list;
        };
    };
}

var transducer =
    composeRight( mapReducer(add1), filterReducer(isOdd) )(listCombination);

var list = [2,5,8,11,14,17,20];

list
.reduce( transducer, [] )
.reduce( sum );
// 48
```

```
function mapReducer(mappingFn) {
    return function toCombine(combineFn){
        return function reducer(list,v){
            return combineFn( list, mappingFn(v) );
        };
    };
}

function filterReducer(predicateFn) {
    return function toCombine(combineFn){
        return function reducer(list,v){
            if (predicateFn(v)) return combineFn( list, v );
            return list;
        };
    };
}

var transducer =
    composeRight( mapReducer(add1), filterReducer(isOdd) )(sum);

var list = [2,5,8,11,14,17,20];

list
.reduce( transducer, 0 )
// 48
```

RECAP:

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

KYLE SIMPSON @GETIFY

FUNCTIONAL-LIGHT JAVASCRIPT