Introduction to Functional Programming

Emily Seibert

Hello!

Tools

- https://github.com/emilyseibert/intro-to-functional-programming
- https://codepad.remoteinterview.io
- Each other!

What is Functional Programming?

A way of thinking about and structuring our code

Why learn this stuff anyways?

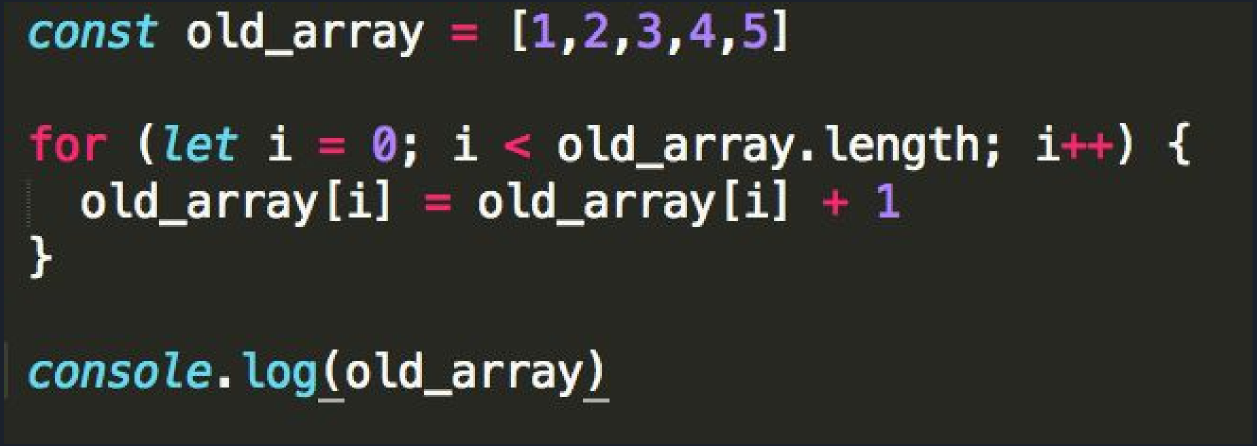
- It cleans up your OO code
- It’s more durable & easier to test
- Parallel processing!!
- It’s making it’s way into front-end development
- Makes you a stronger, more diverse programmer!

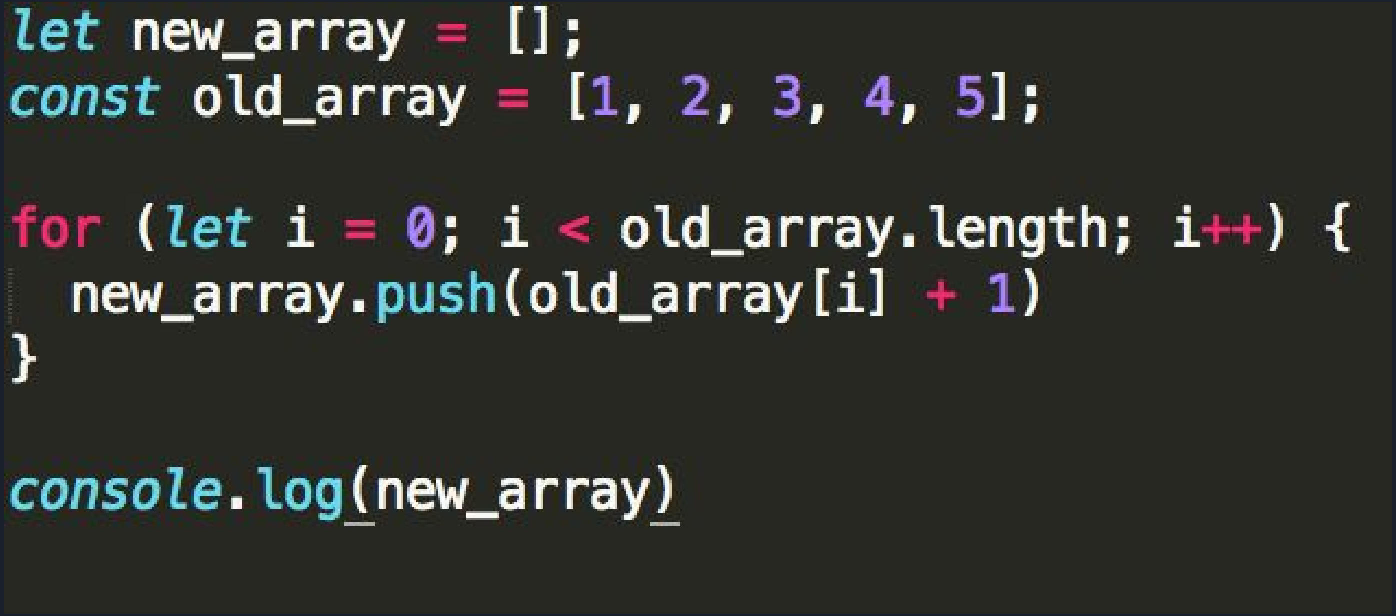
Functional Programming Commandments

1. Your functions must be pure
2. Your methods should err on the side of recursion
3. Your data and code should be encapsulated by closures
4. AND ABOVE ALL ELSE, if it’s mutating it’s not functional.

Languages

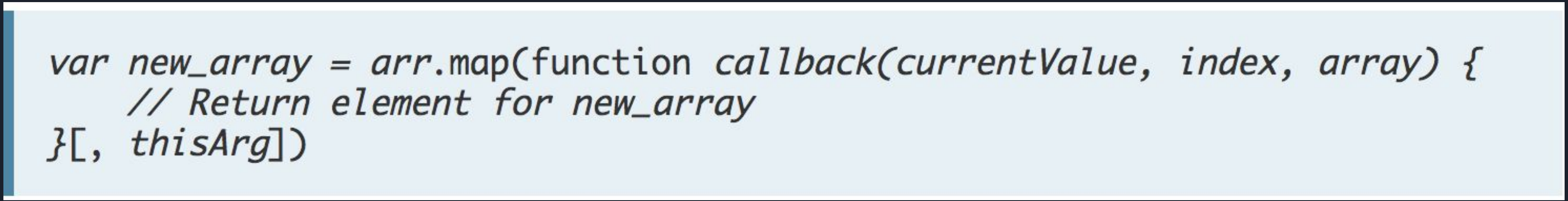
JS


```
const old_array = [1,2,3,4,5]

for (let i = 0; i < old_array.length; i++) {
  old_array[i] = old_array[i] + 1
}

console.log(old_array)
```



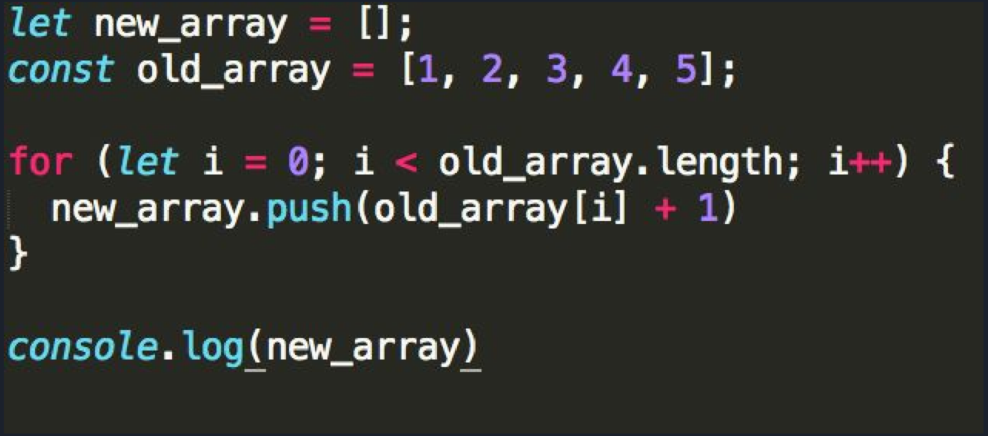
```
let new_array = [];
const old_array = [1, 2, 3, 4, 5];

for (let i = 0; i < old_array.length; i++) {
  new_array.push(old_array[i] + 1)
}

console.log(new_array)
```

JS maps

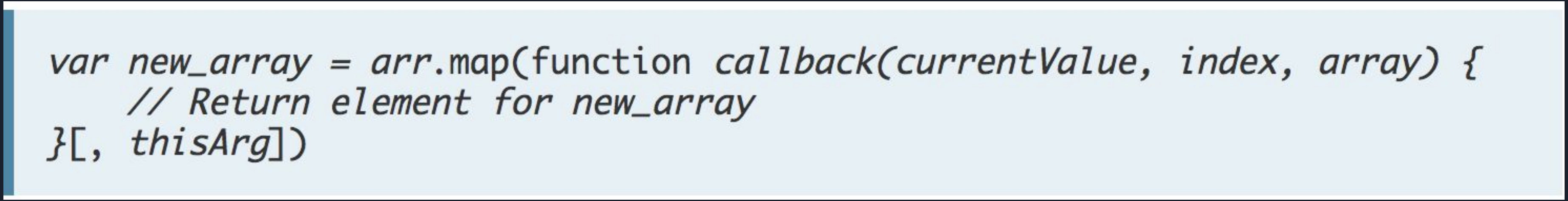


```
var new_array = arr.map(function callback(currentValue, index, array) {
    // Return element for new_array
}[, thisArg])
```

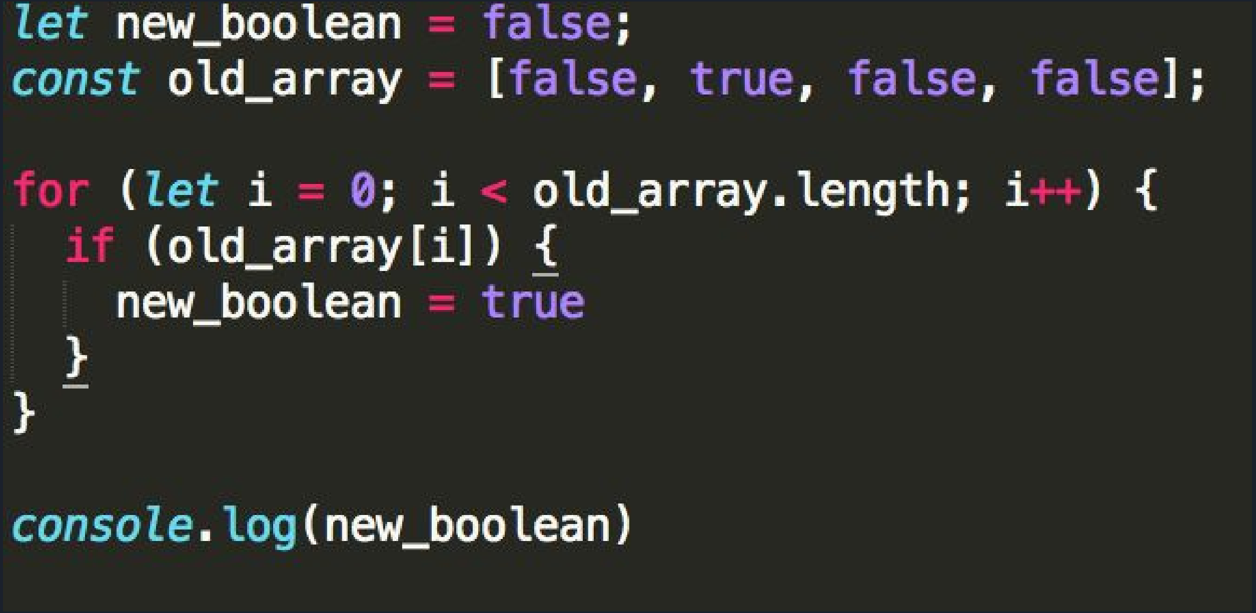
Problem 1 (P1)



```
let new_array = [];
const old_array = [1, 2, 3, 4, 5];

for (let i = 0; i < old_array.length; i++) {
  new_array.push(old_array[i] + 1)
}

console.log(new_array)
```

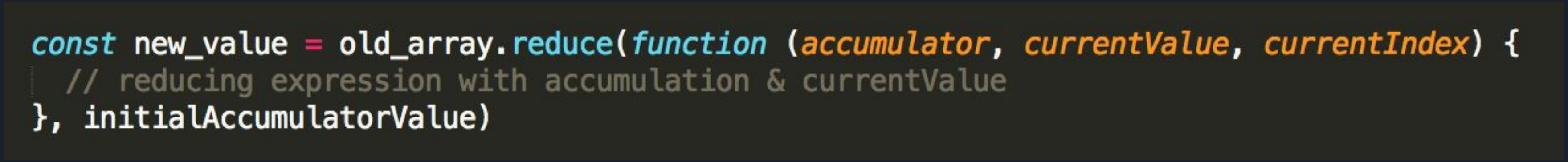


```
var new_array = arr.map(function callback(currentValue, index, array) {
    // Return element for new_array
}[, thisArg])
```



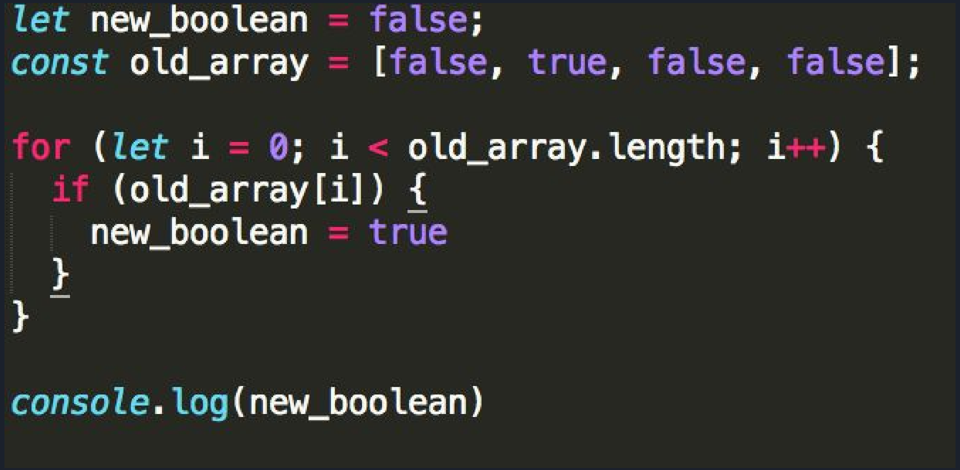
```
let new_boolean = false;
const old_array = [false, true, false, false];

for (let i = 0; i < old_array.length; i++) {
  if (old_array[i]) {
    new_boolean = true
  }
}

console.log(new_boolean)
```

JS reduce

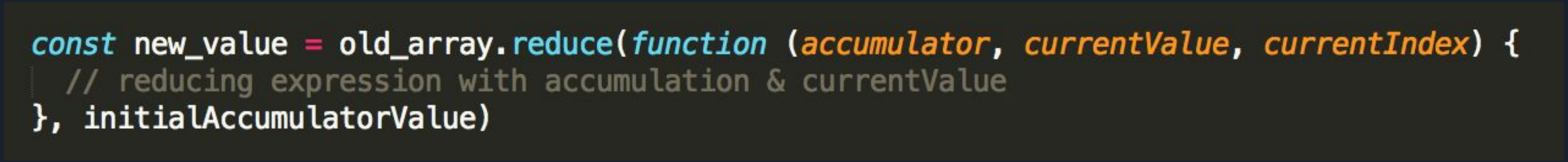


```
const new_value = old_array.reduce(function (accumulator, currentValue, currentIndex) {
  // reducing expression with accumulation & currentValue
}, initialAccumulatorValue)
```

P2



```
let new_boolean = false;
const old_array = [false, true, false, false];

for (let i = 0; i < old_array.length; i++) {
  if (old_array[i]) {
    new_boolean = true
  }
}

console.log(new_boolean)
```



```
const new_value = old_array.reduce(function (accumulator, currentValue, currentIndex) {
  // reducing expression with accumulation & currentValue
}, initialAccumulatorValue)
```

Let’s look at those commandments...

1. **Your functions must be pure** => Where are our pure functions?
2. *Your methods should mostly be recursive*
3. Your data and code should be encapsulated by closures.
4. **And above all else, nothing is mutable**. => Creating new arrays instead of mutating others! Preserve the data!

Lisp?

Basically, inner () first and work your way to the outer ()...

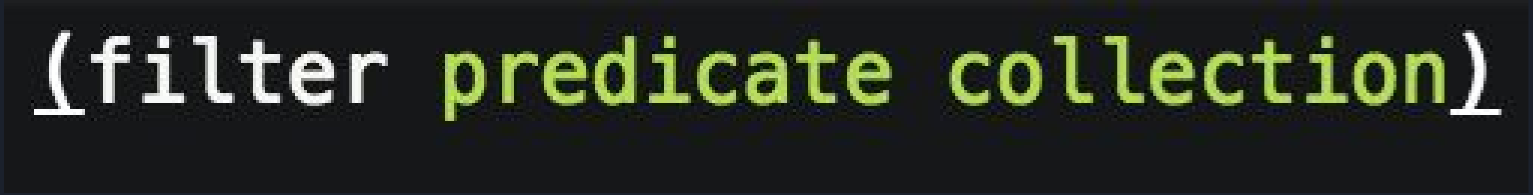
The first element in () is a function.

```
(+ 1 2)
```

(- (+ 1 6) 5)

Defining functions and variables

```
(defn foo [a b c]
      (* a b c))



(def variable “I’m a string!”)
```



```
(filter predicate collection)
```

P3, P4, P5

P3) Write a filter that takes a collection [1, 2, 3, 1] and returns only the elements that equal one

P4) Use the Clojure Docs online to write a predicate that only returns even numbers. Hint: Helper functions are in abundance with Clojure!

P5) Solve P1 and P2 with a Clojure map or reduce function (or both!)

Closures in Clojure

Closure

a function that has access to some named value/variable outside its own scope, so from a higher scope surrounding the function when it was created

Pure Function Currying Madness

```
(defn messenger-builder [greeting]
  (fn [who] (println greeting who)))
```

Pure Function Currying Madness

```
(defn messenger-builder [greeting]
  (fn [who] (println greeting who)))

(def hello-er (messenger-builder
"Hello"))

 (def hello-er
      (fn [who]
           (println “Hello” who)))
```

Pure Function Currying Madness

```
(defn messenger-builder [greeting]
  (fn [who] (println greeting who)))

(def hello-er
     (messenger-builder “Hello”))


(hello-er “world!”)
```

P6 & P7

P6) Use the same structure as the Hello World example to create a closure that adds two numbers together. HINT! Your “def” variable could be a function that always adds a specific number to a parameter

P7) Write a currying function in JS! Translate your work from P6 into JavaScript.

Questions??