Functional Programming in *JavaScript*

Cassidy Williams

@cassidoo

What is functional programming?

"The mustachioed hipster of programming paradigms"

Smashing Magazine

It produces ***abstraction*** through clever ways of combining functions.

"Functional programming [is] a paradigm that forces us to make the complex parts of our system explicit, and that's an important guideline when writing software."

José Valim, creator of Elixir

There are two things you need to know to understand functional programming.

Data is

Immutable

If you want to change data, like an array of data, you return a new array with the changes, not the original.

Functions are

Stateless

Functions act as if for the first time, every time!

3 Best Practices

1. Your functions should accept at least 1 argument

2. Your functions should either return data, or another function

3. Don't use loops!

Quick Example

The OOP Way

```
class Student {
  constructor(name, gpa) {
    this.name = name;
    this.gpa = gap;
  }

  getGPA() {
    return this.gpa;
  }

  changeGPA(amount) {
    return this.gpa + amount;
  }
}
```

```
let phil = new Student('Phil Eaglesworth', 3.95);
```

```
let students = [
    new Student('Phil Eaglesworth', 3.95),
    new Student('Cassidy Williams', 4.0),
    new Student('Joe Randy', 2.2) ];
```

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

The Functional Way

```
let students = [
  ['Phil Eaglesworth', 3.95],
  ['Cassidy Williams', 4.0],
  ['Joe Randy', 2.2],
];
```

```
let newStudents = students.map(function(s) {
  return [s[0], s[1] + .1];
});
```

```
function changeGPAs(students) {
  return students.map(student => changeGPA(student, .1))
}

function changeGPA(student, amount) {
  return student[1] + amount
}
```

Debugging Functional Programming

Yet another quick example!

```
let count = 0;

function increment() {
  if (count !== 4) count += 1;
  else count += 2;

  return count
}
```

```
function pureIncrement(count) {
  if (count !== 4) return count + 1;
  else return count + 2;
}
```

It's a lot like math

oh no

```
(6 * 9) / ((4 + 2) + (4 * 3))
```

```
(define (mathexample)
   (/
      (* 6 9)
      (+
         (+ 2 4)
         (* 4 3)
      )
   )
)
```

There are languages made specifically for this

- Lisp
- Elixir
- Haskell
- Scala
- Clojure

J A V A S C R I P T

(we're about to code, get your laptops ready)

```
function add(a, b) {
  return a + b;
}
```

Write a function that adds from two invocations.

```
addf(3)(4)
```

Write a function that adds from two invocations.

```
function addf(x) {
    return function (y) {
        return add(x, y);
    };
}
```

Write a function that takes in a function and an argument, and returns a function that can take a second argument.

```
curry(add, 9)(3)
```

Write a function that takes in a function and an argument, and returns a function that can take a second argument.

```
function curry(fun, a) {
  return function(b) {
    return fun(a, b)
  };
}
```

You just learned currying!

Write a function that takes a binary function and makes it callable with 2 invocations.

```
liftf(add)(2)(3)
```

Write a function that takes a binary function and makes it callable with 2 invocations.

```
function liftf(fun) {
  return function(a) {
    return function(b) {
      return fun(a, b);
    };
  };
}
```

Last one!

Using the functions we've written so far, write a function **increment**!

```
let increment = curry(add, 1);


> increment(5)
6
```

Using the functions we've written so far, write a function **increment**!

```
let increment1 = addf(1);
let increment2 = liftf(add)(1);
```

Wasn't this *FUN?*

- Functions can be broken down into simpler and smaller chunks that are easier to read
- Programs can be easier to debug due to its modularity
- It is very fun

Thank you!

@cassidoo