

# Functional Programming

Mohd Faiz Hasim

faiz.hasim@servicerocket.com • faizhasim@gmail.com

Some Info

- Also available as [PDF](#), [EPUB](#) and [MOBI](#) formats.
- Hosted at [Github](#).
- Mistakes? Improvements? Make me a pull request.

## What is Functional Programming???

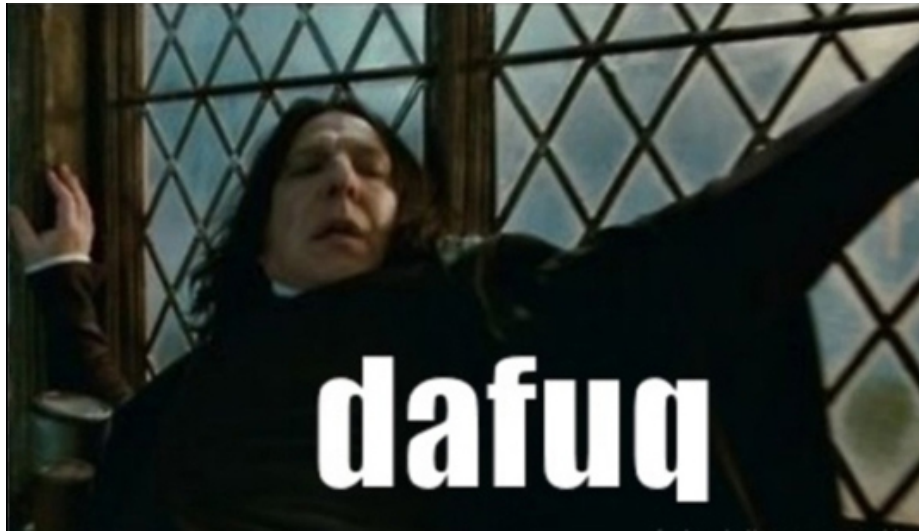
### Computation as Functions

- Lambda Calculus
- Evaluating functions
- Avoid mutability
- Promotes declarative programming

---

### Lambda Calculus - From Wikipedia

- $\text{sumOfSquares}(x,y) = (x \times x) + (y \times y)$
- $(x,y) \rightarrow (x \times x) + (y \times y)$
- $((x,y) \rightarrow (x \times x) + (y \times y))(5,2)$
- $((((x,y) \rightarrow (x \times x) + (y \times y))(5))(2)$



## Imperative vs Functional

### Example data

```
var subscribersOfSocialMedias = [{  
  serviceName: 'facebook',  
  count: 35433,  
  hasOfficialSupport: true  
}, {  
  serviceName: 'twitter',  
  count: 25433,  
  hasOfficialSupport: true  
}, {  
  serviceName: 'instagram',  
  count: 2348,  
  hasOfficialSupport: false  
}];
```

Should give total count of 63214.

---

```
var total = 0;  
for (var i = 0; i < subscribersOfSocialMedias.length; i++) {  
  total += subscribersOfSocialMedias[i].count;  
}
```

```
}  
  
console.log(total);
```

Imperative approach...

---

```
var subscriberCount = function(subscriberInfo) {  
    return subscriberInfo.count;  
}  
  
var accumulate = function(previousValue, currentValue) {  
    return previousValue + currentValue;  
}  
  
var total = subscribersOfSocialMedias  
    .map(subscriberCount)  
    .reduce(accumulate);  
  
console.log(total);
```

Functional approach...

---

```
var withOfficialSupport = function(officiallySupported) {  
    return function(subscriberInfo) {  
        return subscriberInfo.hasOfficialSupport === officiallySupported;  
    }  
}  
  
var total = subscribersOfSocialMedias  
    .filter(withOfficialSupport(true))  
    .map(subscriberCount)  
    .reduce(accumulate);
```

And, to filter by the officially supported social medias.

---

Exact same code with [CoffeeScript](#):



```
subscriberCount = (subscriberInfo) -> subscriberInfo.count

withOfficialSupport = (officiallySupported) ->
  (subscriberInfo) ->
    subscriberInfo.hasOfficialSupport is officiallySupported

total = subscribersOfSocialMedias
  .filter (withOfficialSupport true)
  .map subscriberCount
  .reduce ((a,b) -> a + b)
```

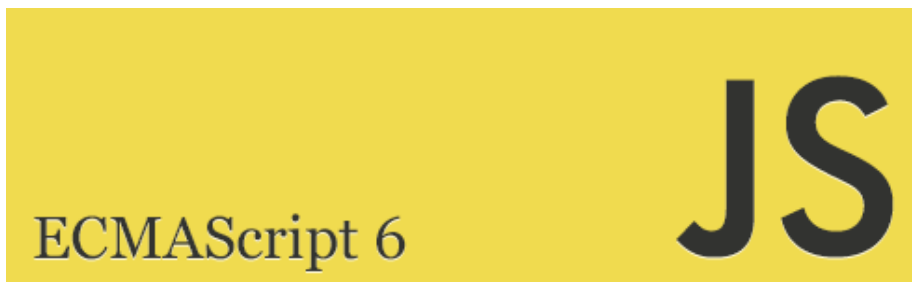
---

Wait, what about ECMAScript 6?

```
var subscriberCount = (subscriberInfo) => subscriberInfo.count

var accumulate = (previousValue, currentValue) => previousValue + currentValue

var withOfficialSupport = (officiallySupported) => (subscriberInfo) => {
  return subscriberInfo.hasOfficialSupport === officiallySupported
}
```



```
}
```

```
let total = subscribersOfSocialMedias
    .filter(withOfficialSupport(true))
    .map(subscriberCount)
    .reduce((a,b) => a + b)
```

CoffeeScript influenced TC-39 decision making.

- 
- [Array#reduce](#)
  - [Array#map](#)
  - [Array#filter](#)
  - [Array#forEach](#)

---

## Libraries that Promotes Functional

### [Underscore.JS](#)

- Very clean API and source code.
- Older established framework and products uses this (eg Confluence).
- My recommendation:



Figure 1: All modern browsers ( < IE 9)



Figure 2: Should we continue?

### Lo-Dash

- Very similar to Underscore.JS, except more performant.
  - Roadmap: Lazy sequence/stream.
  - Supports compatibility with Underscore API.
  - My recommendation:
- 

### Lazy.js

- Just like underscore, but not compatible at all.
  - Key feature: Lazy evaluation on collections or stream.
  - My recommendation:
- 

### Allong.es

- Facilitate using functions as first-class values.
- Fundementally build from `curry`-ing and partial applications.
- My recommendation: If you think you need it, use it.



Figure 3: I like Curry... do you? Let's talk curry.

## Partial Application and Currying

---

### Recommended reads

- [Chapter 5 - Higher-Order Functions](#) of [Eloquent Javascript](#) - by Marijn Haverbeke
  - [JavaScript Allongé](#) - by Reg Braithwaite
- 

### Curry

#### Revisiting previous equations

- $\text{sumOfSquares}(x,y) = (x \times x) + (y \times y)$
  - $(x,y) \rightarrow (x \times x) + (y \times y)$
  - $((x,y) \rightarrow (x \times x) + (y \times y))(5,2)$
  - $((((x,y) \rightarrow (x \times x) + (y \times y))(5))(2))$
- 

$\text{sumOfSquares}(x,y) = (x \times x) + (y \times y)$

```
var sumOfSquares = function(x, y) {  
  return (x * x) + (y * y);  
}
```

---



$(x,y) \rightarrow (x \times x) + (y \times y)$

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

Just a lambda (anonymous function)

---

### Currying?

Turning  $((x,y) \rightarrow (x \times x) + (y \times y))(5,2)$  into  $((x,y) \rightarrow (x \times x) + (y \times y))(5))(2)$

Mathematically, if  $f(x,y) = (x \times x) + (y \times y)$ , then:

$h(x) = y \rightarrow f(x,y)$

---

### Partial application?

$h(x) = y \rightarrow f(x,y)$

$h(x)$  is a partial application of the full application.

---

### Curry + Partial Application

Using `allong.es` at [allong.es/try](https://allong.es/try):

```
var curry = allong.es.curry;  
  
var giveGreetingFrom = curry(function(greeter, targetPerson) {  
  return greeter + ' is saying "hi" to ' + targetPerson;  
})  
  
var giveGreetingFromTom = giveGreetingFrom('Tom');  
  
console.log(giveGreetingFromTom);  
// Will return unary partial application function
```

```

console.log(giveGreetingFromTom('Bill'));
// Tom is saying "hi" to Bill

console.log(giveGreetingFrom('Tom', 'Bill'));
// Tom is saying "hi" to Bill

console.log(giveGreetingFrom('Tom')('Bill'));
// Tom is saying "hi" to Bill

```

## Useful functions allong.es

---

Shamelessly taken from [allong.es/try](http://allong.es/try).

### Fluent

```

var fluent = allong.es.fluent;

Role = function () {};

Role.prototype.set = fluent( function (property, name) {
  this[property] = name
});

var doomed = new Role()
  .set('name', "Fredo")
  .set('relationship', 'brother')
  .set('parts', ['I', 'II']);

doomed
//=> {"name":"Fredo","relationship":"brother","parts":["I","II"]}

```

### Once

```

var once = allong.es.once;

var message = once( function () { return "Hello, it's me"; });

message()
//=> "Hello, it's me"
message()

```

```
    //=> undefined
message()
    //=> undefined
message()
    //=> undefined
```

Also available with underscore.

## Trampolining

Stack-friendly recursion.

```
var trampoline = allong.es.trampoline,
    tailCall = allong.es.tailCall;

function factorial (n) {
  var _factorial = trampoline( function myself (acc, n) {
    return n > 0
      ? tailCall(myself, acc * n, n - 1)
      : acc
  });

  return _factorial(1, n);
};

factorial(10);
//=> 3628800
```