# 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

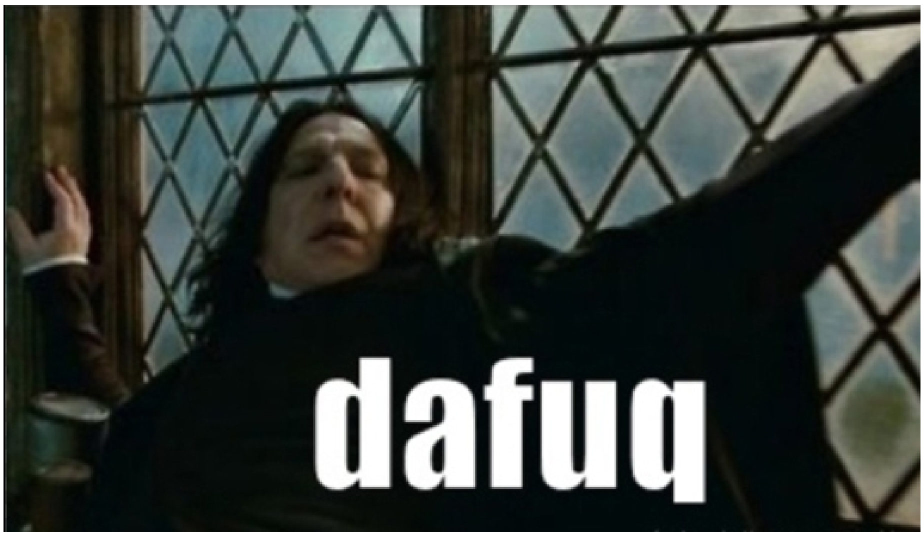
---

#### Lambda Calculus - From Wikipedia

- sumOfSquares(x,y) = (x × x) + (y × y)
- (x,y)   (x × x) + (y × y)
- ((x,y)   (x × x) + (y × y))(5,2)
- (((x,y)   (x × x) + (y × y))(5))(2)

## Why Functional Programming in JS?

---



1. Complexity of States

---

2. Play nice - Now & Future

```
requestBillingDetails(allVendors)
  .then(compose(extractContacts, latePayment))
  .then(sendEmailNotification)
  .catch(ConnectionException, handleConnectionError)
  .catch(handleGenericError);
```

Promise spec (pipelining)

---

3. Scalability and Reusability

- Web workers.
- Function: Do one thing well, without side-effects.

---

4. Still play nice with existing stuff

Figure 1: It's going to hurt now and tomorrow...

- Plain old Javascript object

```
var Employee = new function(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}

Employee.prototype.fullName = fluent(function(){
  return this.firstName + " " + this.lastName;
});

Employee.prototype.applyLeave = fluent(function(from, to) {
  var leaveInfo = LeaveBuilder
    .by(this)
    .from(from)
    .to(to)
    .build();

  LeaveSystem
    .submit(leaveInfo)
    .then(notifyManager());
});
```

## Imperative vs Functional

### Example data

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

Should give total count of `63214`.

---

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

console.log(total);
```

Imperative approach…

---

```
var subsriberCount = function(subsriberInfo) {
  return subsriberInfo.count;
}

var accumulate = function(previousValue, currentValue) {
  return previousValue + currentValue;
}

var total = subsribersOfSocialMedias
              .map(subsriberCount)
              .reduce(accumulate);

console.log(total);
```

Functional approach…

---

```
var withOfficialSupport = function(officiallySupported) {
  return function(subsriberInfo) {
    return subsriberInfo.hasOfficalSupport === officiallySupported;
  }
}

var total = subsribersOfSocialMedias
              .filter(withOfficialSupport(true))
              .map(subsriberCount)
              .reduce(accumulate);
```

And, to filter by the officially supported social medias.

---

Exact same code with CoffeeScript:

```
subsriberCount = (subsriberInfo) -> subsriberInfo.count

withOfficialSupport = (officiallySupported) ->
    (subsriberInfo) ->
        subsriberInfo.hasOfficalSupport is officiallySupported


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

---

Wait, what about ECMAScript 6?

```
var subsriberCount = (subsriberInfo) => subsriberInfo.count

var withOfficialSupport = (officiallySupported) => (subsriberInfo) => {
  return subsriberInfo.hasOfficalSupport === officiallySupported
```

```
}

let total = subsribersOfSocialMedias
            .filter(withOfficialSupport(true))
            .map(subsriberCount)
            .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 2: All modern browsers (  IE 9)

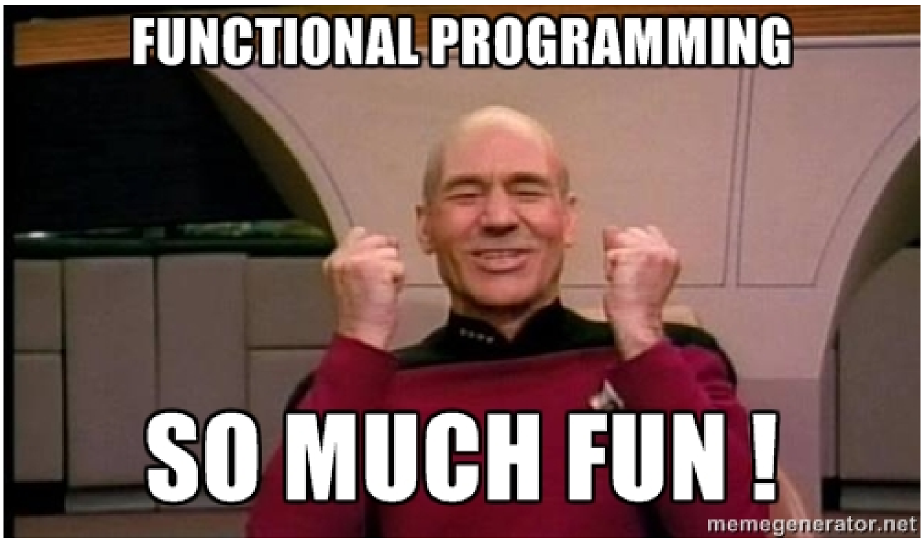


Figure 3: Should we continue?

## 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 4: 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

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

### sumOfSquares(x,y) = (x × x) + (y × y)

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

**(x,y)   (x × x) + (y × y)**

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

Just a lambda (anonymous function)

---

**Currying?**

- Turning `((x,y)   (x × x) + (y × y))(5,2)` into `(((x,y)   (x × x) + (y × y))(5))(2)`
- Mathematically, if `f(x,y) = (x × x) + (y × y)`, then:

  h(x) = y   f(x,y)

---

**Partial application?**

`h(x) = y   f(x,y)`

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

---

**Curry + Partial Application**

Using `allong.es` at 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.

### 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
```