Immutability 101

what, why, and how

Hello LambdaConf

twitter: @pauldegoes

email: pauldegoes@hotmail.com

medium: https://medium.com/@peregrinef8

workshop: https://github.com/pauldegoes/presentations/tree/master/lambdaconf-2016

Hello LambdaConf — cats

Immutability — What

Origin: Latin

in (not) - *mutabilis* (to change)

Immutability — What

Software (*functional programming)*

Object or structure whose **state cannot be modified** after it is created

Numbers are immutable

```
var x = 1;

x == 1;    // => true

x + 1;     // => 2

x == 1;    // => true
```

Strings are immutable

```
var x = "foo";

x == "foo";                 // => true

x + "bar";                  // => "foobar"

x == "foo";                 // => true

(x + "bar").substr(2,3);    // "oba"
```

Booleans are immutable

```
var x = true;

x == true;      // => true

x && true;      // => true

x && false;     // => false

x == true;      // => true
```

variables are… mutable!

How can a variable be immutable?

```
var x = true; x;      // => true

x = 1; x;             // => 1

x = “ANYTHING!!"; x; // => “ANYTHING!!”
```

variables are… mutable!

They cant. But values can be!

Immutability — what's in it for me?

Immutable Data Structures are easier to reason about because

1. They operate uniformly

Immutability — what's in it for me? (ex.1.1)

```
var last_weeks_dollars = 6;

var more_dollars = last_weeks_dollars + 2;

more_dollars;         // 8
last_weeks_dollars; // 6
```

Immutability — what's in it for me? (ex.1.2)

```
var last_weeks_dollars = [2,2,2];

var more_dollars = last_weeks_dollars.push(2);

more_dollars;         // ?
last_weeks_dollars; // ?
```

Immutability — what's in it for me? (ex.1.3)

```
var last_weeks_dollars = [2,2,2];

var more_dollars = last_weeks_dollars.push(2);

more_dollars;         // 4
last_weeks_dollars; // [2,2,2,2]
```

Immutability — what's in it for me?

Immutable Data Structures are easier to reason about because

1. They operate uniformly (no surprises)
2. They are what they are (no mutations)

Immutability — getting started

What do I do now?

Immutability — getting started

3 step plan to combat mutants (mutation)!

1. Admit that mutation is evil

Immutability — getting started

3 step plan to combat mutants (mutation)!

1. Admit that mutation is evil
2. Stop creating mutation

Immutability — getting started

3 step plan to combat mutants (mutation)!

1. Admit that mutation is evil
2. Stop creating mutation
3. Create immutable things!

Problem 1 — the immutable list

Create a replacement for the Javascript Array class that:

1. Does not mutate the initializing value (if present)
2. Does not expose mutative methods
3. Does mimic the following methods:

    - push
    - unshift
    - concat
    - length
    - [index]

Immutability — The Immutable List (prb.1.1)

```
function List(array) {
  this.mutable_list = array.slice();

  this.length = this.mutable_list.length;
}

(new List([9, 8])).length // => 2
```

Immutability — The Immutable List (prb.1.2)

```
function List(array) {
  this.mutable_list = (array || []).slice();

  this.length = this.mutable_list.length;

  this.to_array = this.mutable_list;

  this.immutable_list = function() {
    return new List(this.mutable_list);
  }

  this.value_of_index = function(index) {
    return this.mutable_list[index];
  }

  this.append = function(value) {
    return new List(this.mutable_list.concat([value]));
  };

  this.prepend = function(value) {
    return new List([value].concat(this.mutable_list));
  }

  this.concat = function(other_mutable_array) {
    return new List(this.mutable_list.concat(other_mutable_array));
  }
}
```

Immutability — The Immutable List (prb.1.3)

```
(new List([1])).append(2).prepend(0).concat([3,4]).to_array

// => [0,1,2,3,4]
```

Immutability — The Immutable List (prb.1.4)

What does that look like without the immutable list?

Immutability — The Immutable List (prb.1.5)

I’m so glad you asked!

```
var a1 = [1];
var a2 = a1.slice();
a2.push(2);
var a3 = a2.slice();
a3.unshift(0);
var a4 =  a3.concat([3,4]);
a4;

// => [0,1,2,3,4]
```

Problem 2 — the immutable map

Create a replacement for the Javascript Object class that:

1. Does not mutate the initializing value (if present)
2. Does not expose mutative methods
3. Does mimic the following methods:

    - [key] = value
    - [key]

Immutability — The Immutable Map (prb.2.1)

```
function Map(mutable_object) {
  this.shallow_copy = function(mut_obj) {
    var copy = {};

    for (var key in mut_obj) {
      copy[key] = mut_obj[key];
    }

    return copy;
  };

  this.to_object = function() {
    return this.mutable_object;
  };

  this.set = function(key, value)  {
    this.mutable_object[key] = value;
    return new Map(this.mutable_object);
  };

  this.get = function(key)  {
    return this.mutable_object[key];
  };

  this.mutable_object = this.shallow_copy(mutable_object);
};
```

Immutability — The Immutable Map (prb.2.2)

```
new Map({name: "Gollum"}).set("age", 503).set("hasRing", false).to_object

// => {name: "Gollum", age: 503, hasRing: false}
```

Immutability — complications

Immutable Data Structures

are only as reliable

as the data you give them

Immutability — complications

```
var mutable_object = {};
var immutable_list = new List([mutable_object]);

mutable_object["mutating"] = true;

immutable_list.value_of_index(0).mutating;

// => true
```

Immutability — complications

```
var mutable_object = {};
var immutable_map = new Map({person: mutable_object})
mutable_object["name"] = "Bubbles"
immutable_map.get("person").name

// => "Bubbles"
```

Immutability — complications

So how do we deal with bad data?

Immutability — complications

It depends…

- Be snobby (never provide bad data)

Immutability — complications

It depends…

- Be snobby (never provide bad data)
- Be meticulous (sanitize the data)
  - Improved object cloning
  - Value type checking

Problem 3 — counting

Given an Array[Person], count all people aged 21 and up

```
var attendees = [
  {name: "tom", age: 30},
  {name: "jane", age: 21},
  {name: "robin", age: 20}
]
```

The old way…

```
var selected = [];

for (var i=0; i < attendees.length; i++) {
  var cur_attendee = attendees[i];
  if (cur_attendee.age >= 21) {
    selected.push(cur_attendee);
  }
}

selected.length; // => 2
```

Immutability — In Real Life (prb.3.2)

The new way…?

```
var selected = new List([]);

. . .
```

The new way…?

```
var selected = new List([]);

for (var i=0; i < attendees.length; i++) {
  var cur_attendee = attendees[i];
  if (cur_attendee.age >= 21) {
    selected = selected.append(cur_attendee)
  }
}

selected.length; // => 2
```

The better way

through **recursion**

The better way

through **recursion**

The invocation of a function from
within the same function
in order to produce a desired result

Immutability — Recursion - what is it?

What is recursion good for?

- Solving the same type of problem on the same type of data for an unknown quantity of data
- Iterative branching
- When large problems are composed of smaller simplistic versions of the larger problem
- Iterating (less efficiently)

Problem 4— creating a for_each method without a for loop

Without using native Javascript loops, create a function that allows an operation to be performed on each element of an array

```
for_each([1,2,3], function(el) {
  console.log(el)
})

// logs 1, 2, and 3 to the console
```

Immutability — Recursion (prb.4.1)

```
function for_each(array, fn) {
  if (array.length > 0) {
    fn(array[0]);
    for_each(array.slice(1, array.length), fn);
  }
}
```

Problem 5— generating a fibonacci sequence

Given n iterations, find the fibonacci number associated with n

First 10 fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

Immutability — Recursion (prb.5.1)

```
function fib(iterations, n, prev_n) {

}
```

OR

```
function fib2(iterations) {

}
```

Immutability — Recursion (prb.5.1)

```
function fib(iterations, n, prev_n) {
  n        = n || 0;
  prev_n   = prev_n || 0;

  if (iterations > 0) {
    var cur_n = (n == 0) ? 1 : (n + prev_n);

    return fib(iterations -1, cur_n, n)
  } else {
    return n
  }
}

[fib(0), fib(1), fib(2), fib(3), fib(4), fib(5)]

// => [0, 1, 1, 2, 3, 5]
```

Problem 6— making our List more useful

Add the following methods to our List class:

- as_list
- head
- tail
- fold

Immutability — Recursion (prb.6.1.1)

as_list:

takes a mutable array and returns an immutable list

```
this.as_list = function(mutable_array) {

};
```

Immutability — Recursion (prb.6.1.2)

as_list:

takes a mutable array and returns an immutable list

```
this.as_list = function(mutable_array) {
  return new List(mutable_array);
};
```

Immutability — Recursion (prb.6.2.1)

head:

returns the first entry in the mutable list or null

```
this.head = ;
```

Immutability — Recursion (prb.6.2.2)

head:

returns the first entry in the mutable list or null

```
this.head = this.mutable_list[0] || null;
```

Immutability — Recursion (prb.6.3.1)

tail:

returns a List with all members of the current set except for the first element

```
this.tail = function() {

};
```

Immutability — Recursion (prb.6.3.2)

tail:

returns a List with all members of the current set except for the first element

```
this.tail = function() {
  return this.as_list(
    this.mutable_list.slice(1, this.length)
  )
};
```

fold:

takes an accumulator and a function that takes an accumulator and a single element

```
this.fold = function(accumulator, fn) {

};
```

Immutability — Recursion (prb.6.4.2)

fold:

takes an accumulator and a function that takes an accumulator

and a single element

```
this.fold = function(accumulator, fn) {
  if (this.head) {
    return this.tail().fold(fn(accumulator, this.head), fn);
  } else {
    return accumulator;
  }
};
```

Immutability — In Real Life (prb.4.1)

The new way!

```
new List(attendees).fold(
  new List,
  function(accum, el) {
    return el.age >= 20 ? accum.append(el) : accum;
  }
).length

// => 2
```

Immutability — further study

Where do I go from here?

Immutability — furthering fold

Take fold further

- implement all of the following functional methods through fold
  - map
  - filter
  - any
  - all
  - take
  - take_while

Immutability — improved cloning

Use recursion to create a better deep cloning method for initialization of your List and Map

Immutability — libraries

Check our the following immutable libraries

- Immutable.js
  - https://facebook.github.io/immutable-js/
- Mori
  - http://swannodette.github.io/mori/
- Seamless-immutable
  - https://github.com/rtfeldman/seamless-immutable

Goodbye LambdaConf

twitter: @pauldegoes

email: pauldegoes@hotmail.com

medium: https://medium.com/@peregrinef8

workshop: https://github.com/pauldegoes/presentations/tree/master/lambdaconf-2016