Why Functional Programming matters?

Hello, World!

- **Diego Freniche** (@dfreniche)
- 100% remote dev @ Mobile Jazz (iOS / Android)
- SCJP 1.5, SCJP 1.6, SCWCD 1.5, SCBCD 1.3, Itil Foundations [1]

[1] CV: `git clone http://www.github.com/dfreniche/cv`

F.P.

- invented in the 50's (1958)
- only ALGOL is older
- derived from the *Lambda Calculi* (developed in the 30's)

LISP in College

- Lost In Stupid Parentheses
- Lots of Irritating Superfluous Parentheses
- ...

A word (or two) on LISP

- simplest syntax for ANY programming language
- Code:

```
(+ 3 2)
```

- List of data:

```
'("hello", "world")
```

A word (or two) on LISP

- **homoiconicity**: "the structure of program code is represented faithfully and directly in a standard data structure"
- we have direct access to the compiler's AST
- Much power. So cool. Such compiler. Wow

Eval: treat data like code, code like data

- Eval: treat data like code, code like data

```
CL-USER> '(+ 3 4)
(+ 3 4)
CL-USER> (+ 3 4)
7
CL-USER> (eval '(+ 3 4))
7
```

http://learnlispthehardway.org/try-lisp/

Ideas piooneered by LISP

- tree data structures
- automatic storage management
- dynamic typing
- conditionals
- higher-order functions
- recursion
- the self-hosting compiler

Greenspun's tenth rule [2]

Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

---

[2] https://en.wikipedia.org/wiki/Greenspun%27s*tenth*rule

F.P. is back!

- Closures & High Order Funcions in Swift. Inmutability. Optionals.
- Blocks in Objective-C to mimic Closures.
- Scala
- C# / F#
- JavaScript?

Being a Functional Language vs. having Functional Constructs

There's no accepted definition of functional programming language.

–

If you define functional language as the language that supports first class functions and lambdas, then yes, JavaScript is a functional language.[3]

---

[3] http://stackoverflow.com/questions/3962604/is-javascript-a-functional-programming-language

Functional Programming

... functional programming is a programming paradigm ... that treats computation as the **evaluation of mathematical functions** and **avoids changing-state and mutable data**. It **is a declarative programming paradigm**,

Functional Programming

... the output value of a function **depends only on the arguments that are input to the function**, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time **eliminating side effects**

https://en.wikipedia.org/wiki/Functional_programming

OK, Something practical, please?

Inmutability, avoiding state, don't shoot yourself in the foot

- Benefits of value types and immutability
  - compiler can optimize your code
  - eats more memory, optimizes *programmer's* time

Inmutability

- Diego's dumb rule

  Declare everything as a constant, let compiler warn you when you're changing something. That's a variable.

In Java:

```
final String s = "";

s = "other thing"; // nope
```

Inmutability

- nice to avoid stupid mistakes

```
public void downloadImage(String imageUrl) {
    imageUrl += "?something=10";    // don't do this
}

public void downloadImageInmutable(final String imageUrl) {
    imageUrl += "?something=10";    // now you can't do stupid & ugly things
}
```

Inmutability

- why so final?
- closures, anyone?

Inmutability

```
protected void onCreate(Bundle icicle) {
    // more code here, including a call super antipattern...

    final Person diego;
    button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            // Perform action on click

            // use Person object
            diego.name = "Diego";           // OK
            diego = new Person("Groucho");  // nope
        }
    });
}
```

Use nice annotations (if possible)

- annotations war

```
import com.sun.istack.internal.NotNull;

...

public void downloadImage(@NotNull String imageUrl) {
    imageUrl += "?something=10";    // don't do this
}

public void downloadImageInmutable(@NotNull final String imageUrl) {
    imageUrl += "?something=10";    // now you can't do stupid & ugly things
}
```

Why you hate FP

- Functional programming is declarative
- say *what you want*, not *micromanage how to do it*
- you're already doing it!
  - SQL
  - CSS
  - Regular expressions

High order functions

- Functions that take functions as parameters
- Map, Filter, Reduce, Flatmap
- *Let's do something with these fruits*

Map

- "Do *THIS* to every single element in the list"
  - **add 1** to every element in **this list**
  - **takes** a function (add one) & a list
  - **returns** a list
- *with a basket of fruit: peel every fruit in this basket*
- think of **SQL update**

Map example

```
let numbers = [1, 2, 3, 4, 5, 6]

numbers.map { (e: Int) -> Int in
    return e + 1
}
```

Reduce

- **takes** a function (add) & a list
- returns just **one** element
- *make multi-fruit juice*
- think of AVG function in SQL

Reduce example

```
let numbers = [1, 2, 3, 4, 5, 6]

numbers.reduce(0, combine: +) --> 21
```

Filter

- "I only want oranges from that basket"
- SQL select WHERE clause

Filter example

```
let numbers = [1, 2, 3, 4, 5, 6]

let result = numbers.filter({$0 % 2 == 0})

result --> [2, 4, 6]
```

Flatmap

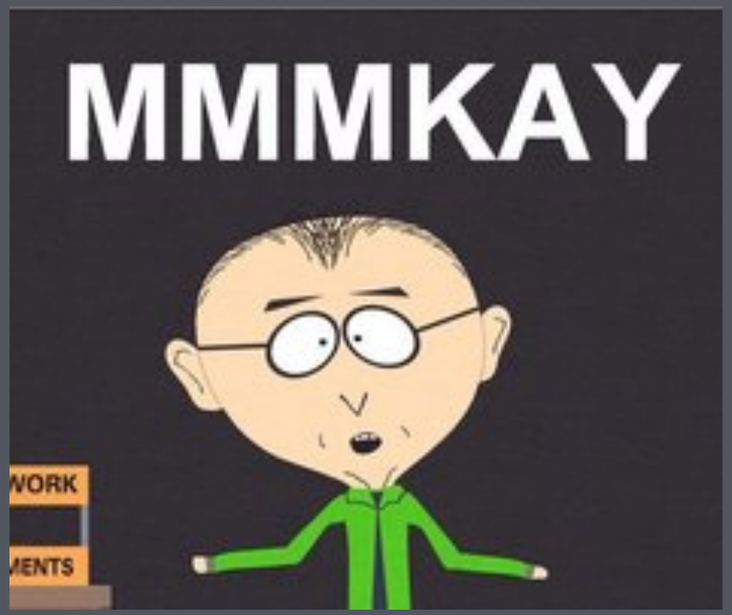
- Like map, but also "flattens" the list
- *some of the fruits in the basket are wrapped with paper*

Flatmap example

```
// how do you add 2 to all the numbers in this array?

let fa2 = [[1,2],[3],[4,5,6]]

let fa2m = fa2.flatMap({$0}).map({$0 + 2})
fa2m
```



So what?

- you don't need to be an athlete to run and be healthier
- you don't need to be "purist" to benefit from some FP goodness

So what?

- use inmutable structures where possible
- in Java, for example, final everything
- in Swift, prefer **structs** vs classes

So what?

- map / filter / reduce helps sometimes
- not everything is a for loop
- use the best of OOP + the best of FP

Bonus tip!

Classic thinking

We have n tags. Need to show only thre first three.

```
- (NSArray <NSString *> *)firstThreeHashtags {
    NSMutableArray <NSString *> *result = [[NSMutableArray alloc] init];

    for (NSString *hastag in self.hashtags) {
        if (hastag.length > 0) {
            [result addObject:hastag];
        }
        if ([result count] > 2) {
            break;
        }
    }
    return result;
}
```

Tell what you want what you really really want...

```
- (NSArray <NSString *> *)firstThreeHashtags {
    NSUInteger rangeMax = MIN(3, self.hashtags.count);
    return [self.hashtags subarrayWithRange:NSMakeRange(0, rangeMax)];
}
```

Thanks!