Author: Craig Clayton

FUNCTIONAL PROGRAMMING

THANK YOU!!

Sylvia Martinez

Kevin

Serguei Vinnitskii

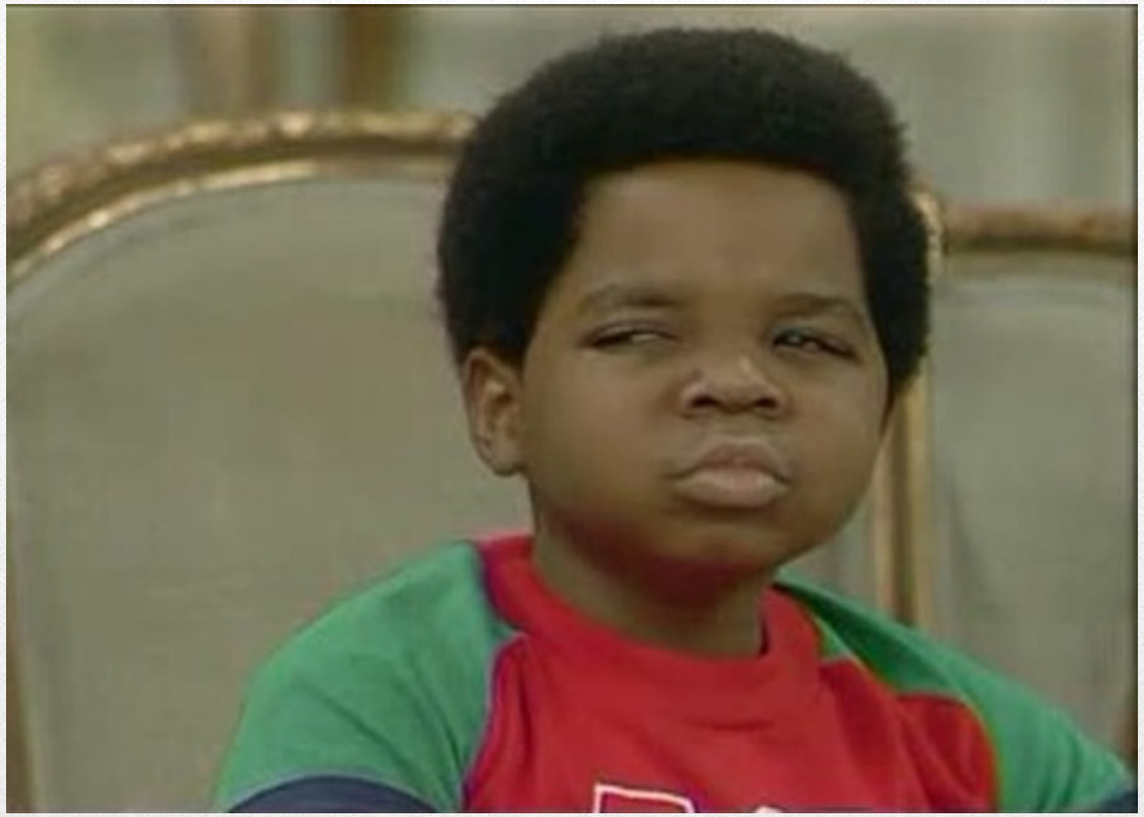
Brian Foshee

Juan Catalan

Scott Steinman

Barry Ezell

SWIFT IS NOT FUNCTIONAL



WHAT WE WILL COVER

1. First Class Functions
2. High Order Functions
3. Chaining
4. Pure Functions
5. Currying

FIRST CLASS FUNCTIONS

Basic Example

```
func basicFunc () -> () -> () {
    func inner() {
        println("Inner function")
    }

    return inner
}

var myFunc =  basicFunc()
myFunc() // “Inner Function”
```

Real World Example

```
func deposit (amount:Float) -> String {
    return String(
        format: "You have deposited $%.2f into your account",
        amount
    )
}

func withdraw (amount:Float) -> String {
    return String(
        format: "You have withdrew $%.2f from your account",
        amount
    )
}
```

Real World Example (cont.)

```
func bankTransaction (type:String) -> (Float) -> (String) {
    if type == "withdraw" {
        return withdraw
    }
    return deposit
}
```

Real World Example (cont.)

```
let morningTransaction = bankTransaction("deposit")
morningTransaction(234.04)
// You have deposited $234.04 into your account

let eveningTransaction = bankTransaction("withdraw")
eveningTransaction(100.00)
// You have withdrawn $100.00 from your account
```

HIGH ORDER FUNCTIONS

Map Function

```
let arrNumbers = [ 1000, 2045, 3500,
                   4099, 5777, 6331, 7000 ]

var formattedNumbers: [String] = []

for number in arrNumbers {
    let formattedNumber =
NSNumberFormatter.localizedStringFromNumber(
        number, numberStyle: .DecimalStyle
    )
    formattedNumbers.append(formattedNumber)
}
```

Map Function

```
let mapFormattedNumbers = arrNumbers.map {
    NSNumberFormatter.localizedStringFromNumber(
        $0, numberStyle: .DecimalStyle
    )
}

mapFormattedNumbers
//1,000, 2,045, 3,500, 4,099, 5,777, 6,331, 7,000
```

Filter Function

```
let filterEvenNumbersOnly = arrNumbers.filter { $0 % 2 == 0 }
filterEvenNumbersOnly
//[1,000, 3,500, 7,000]
```

Reduce Function

```
let totalSum = arrNumbers.reduce(0) { $0 + $1 }
totalSum //29,752
```

CHAINING

Chaining Example

```
struct Person {
    let name: String
    let age: UInt
}

let people = [
    Person(name: "Alice", age: 22),
    Person(name: "Bob", age: 23),
    Person(name: "Mallory", age: 25)
]
```

Chaining Example (cont.)

```
let ageSum = people.map({$0.age}).reduce(0, combine: +)
ageSum // 70
```

Chaining Example (cont.)

```
let morePeople = [
    Person(name: "Alice", age: 22),
    Person(name: "Bob", age: 23),
    Person(name: "Mallory", age: 25),
    Person(name: "Toni", age: 23),
    Person(name: "Zach", age: 21)
]

let namesBeforeJason = morePeople.map({$0.name}).filter {
    name in name.compare("Jason") ==
    NSComparisonResult.OrderedAscending
}
```

Chaining Example (cont.)

```
namesBeforeJason //[“Alice”, “Bob”]
```

Chaining Example (cont.)

```
let namesBeforeJason = morePeople.map({$0.name}).filter {
    name in name.compare("Jason") ==
    NSComparisonResult.OrderedAscending
}
```

Chaining Example (cont.)

```
namesAfterJason //[“Mallory”, “Toni”, “Zach”]
```

PURE FUNCTIONS

Imperative Example

```
var sum = 0.0
var numbers = [10,20,30,40]

func meanImperative() -> Double {
    for number in numbers {
        sum += Double(number)
    }

    let mean = sum / Double(numbers.count)
    return mean
}
```

Imperative Example (cont.)

```
meanImperative() // 25.0
meanImperative() // 50.0
```

Functional Example

```
func meanFunctional(numbers:[Int]) -> Double {
    let sum = numbers.reduce(0, combine: {$0 + $1})
    return Double(sum) / Double(numbers.count)
}
```

Functional Example (cont.)

```
meanFunctional(numbers) // 25.0
meanImperative(numbers) // 25.0
```

CLASS VS STRUCTS

Immutability Example

```
struct StructName {
    var firstName:String
    var lastName:String
}

class ClassName {
    var firstName:String
    var lastName:String

    init(firstName:String, lastName:String) {
        self.firstName = firstName
        self.lastName = lastName
    }
}
```

Immutability Example (cont.)

```
let aStruct = StructName(
    firstName:"Craig",
    lastName:"Clayton"
)
```

```
aStruct.firstName = “Hello"
Cannot assign to ‘firstName’ in ‘aStruct’
```

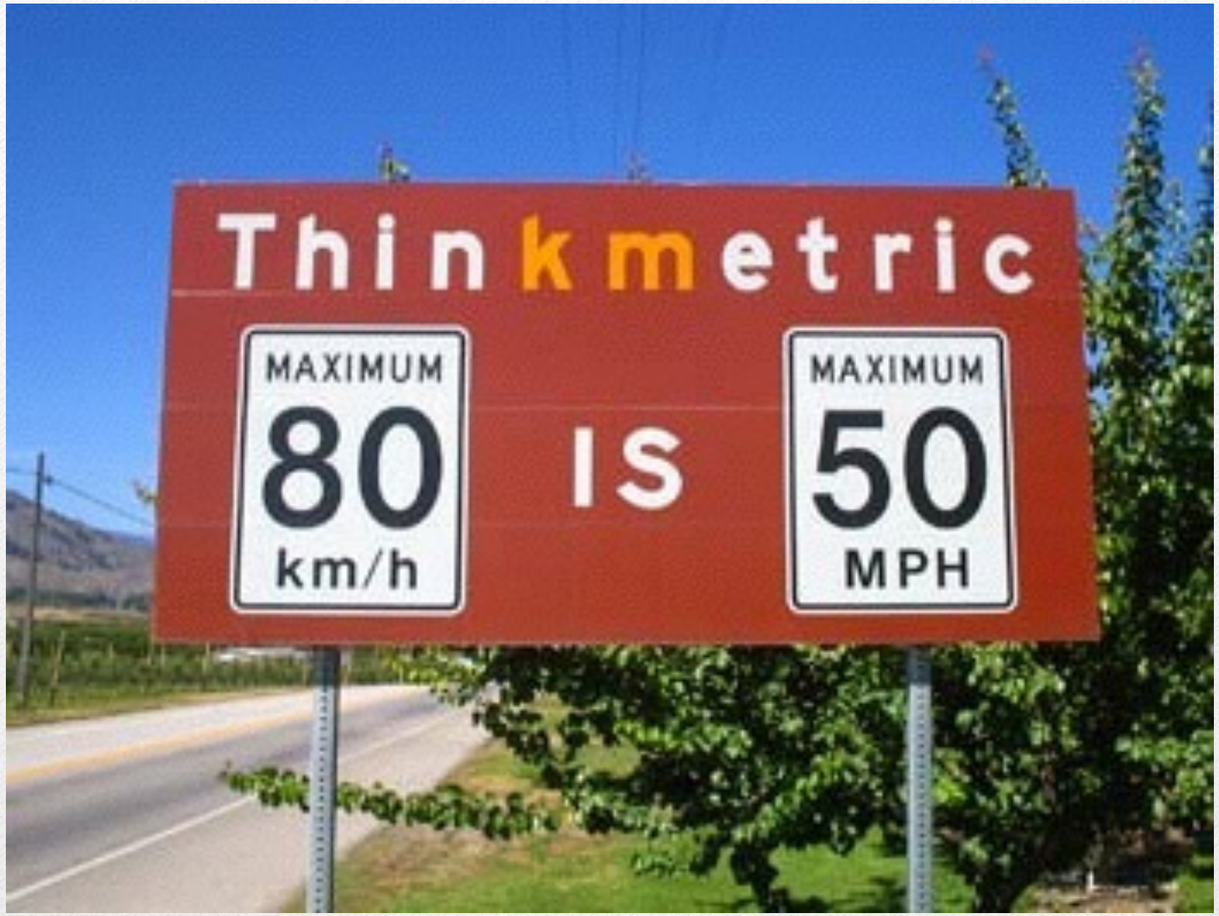
Immutability Example (cont.)

```
let aClass = ClassName(
    firstName:"Neeraj",
    lastName:"Kumar"
)

aClass.firstName = "Hello"
```

CURRYING

Miles to km



Imperative Example

```
func converter(factor:Double, unit:String, toConvert:Double)
-> String {
    return "\(toConvert * factor) \(unit)"
}

let miles2kmConvert  = converter(1.6,  "km", 25) //40.0 km
let pounds2kgConvert = converter(0.45, "kg", 50) //22.5 kg
```

Functional Example

```
func converter(factor:Double)(unit:String)(toConvert:Double)
-> String {
    return "\(toConvert * factor) \(unit)"
}

let miles2kms = converter(1.6)(unit:"km")
miles2kms(toConvert:25)

let pounds2kg = converter(0.45)(unit:"kg")
pounds2kg(toConvert:50)
```

Useful Resources

Websites
http://natashatherobot.com/
https://swiftnews.curated.co/
http://2014.funswiftconf.com/
http://www.objc.io/snippets/

Slides
https://github.com/thedevme/Suncoast-iOS-Meetup

Books
Becoming Functional (Code in Java)
Functional Programming in Swift

Author: Craig Clayton

END! THANK YOU