MOBILE DAY

SWIFT 101

NICOLAS AMEGHINO

@nameghino

JULIO CARRETTONI

@dev_jac

SWIFT INTRODUCTORY WORKSHOP

- Introduction to Swift (15 min)
  - Language fundamentals and origin
  - Syntax and Types
  - Variables and Constants
  - Structs and Enums
  - Classes, Protocols and Extensions
- Workshop (75 min)

CHRIS LATTNER

@clattner_llvm

- LLVM
- Clang
- LLDB
- Swift

Swift

http://swift.org

LANGUAGE FUNDAMENTALS AND ORIGIN

- Safe — type checking, ARC, visibility
- Fast — static dispatch, compile-time optimizations
- Flexible — multi-paradigm language: OOP / Functional / Generic, in constant evolution
- Modern — takes constructions and elements from other languages
- Easy to learn — well known structures, reduced boilerplate, playgrounds (REPL)
- Interoperable — C / Obj-C access w/o additional cost at development time
- Open Source — very active community, language evolution

SYNTAX AND TYPES

```
let interestingNumbers = [
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0
for (kind, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
```

Example taken from: Apple Inc. “The Swift Programming Language.” iBooks. https://itun.es/ar/jEUH0.l

SYNTAX AND TYPES

Swift	Obj-C	C
Int UInt	NSNumber	int
Bool	NSNumber	bool
Float	NSNumber	float
String	NSString	char*
Character	char	char
Array<Tipo>	NSArray	Tipo[]
Dictionary<Clave, Valor>	NSDictionary	
Set<Tipo>	NSSet	

SYNTAX AND TYPES

- Takes elements from multiple languages:
- Trailing Blocks from Ruby

```
array.map { value in
   return value + 1
}
```

- Getters and Setters from C#

```
var a: String { get {} set {} }
```

- Tuples from Python

```
var b = (1, “foo”, Bar())
```

- Optionals from Haskell / Java / C#

```
var c: String? = nil
```

- and more…

VARIABLES AND CONSTANTS

```swift
var hello: String = “world” // Variable

let foo: Int = 10 // Constant

var foo = “bar” // Type inference: String
```

```swift
var foo: Bool { // Computed variables
    get { return textField.visible }
    set { textField.visible = newValue }
}

var bar: Float { // Observers
    willSet { print(“Before: \(newValue)”) }
    didSet { print(“After: \(oldValue)”) }
}
```

VARIABLES AND CONSTANTS

- Value (primitives / structs / enums / tuples)

```
var foo = “world”
var hello = foo
foo = “bar”

// Result:
// hello = “world”
// foo = “bar”
```

- Value (struct)

```
struct Value {
  var property: String;
}
var a = Value(property: “world”)
var b = a
a.property = “hello”

// Result:
// a.property = “hello”
// b.property = “world”
```

- Reference (class)

```
class Reference {
  var property: String;
}
var a = Reference(property: “world”)
var b = a
a.property = “hello”

// Result:
// a.property = “hello”
// b.property = “hello”
```

STRUCTS AND ENUMS

- Primitives, Structures and Enums may have associated functions:

```
struct Structure {
    var foo: Int
    var bar: Int

    init(foo: Int, bar: Int) { // Constructors for Structures are auto-generated!
        self.foo = foo
        self.bar = bar
    }

    func describeMe() -> String {
        return “I’m an structure with values \(foo) and \(bar)”
    }

    mutating func swap() { // Functions changing the value should be marked as mutating
        var tmp = foo
        foo = bar
        bar = tmp
    }
}
```

STRUCTS AND ENUMS

- Enums may have a type ≠ than Int

```
enum Order : String {
    case top = “top”
    case hot = “hot”
    case new = “new”
}

var order = Order(rawValue:“top”)
```

```
enum Order : String {
    case top, hot, new
}
```

- Enums may have associated values

```
enum MUYRedditError : Error {
    case generic(String)
    case wrapped(Error)
    case cocoa(NSError)
}
```

CLASSES AND PROTOCOLS

- Swift supports single inheritance

```
class NSNumber: NSValue {
…
}
```

- A Class, Enum or Struct can implement several protocols

```
enum MUYRedditError : Error {
    case generic(String)
    case wrapped(Error)
}
```

PROTOCOL EXTENSIONS

- Default behavior can be added to a protocol

```
extension Networking {
    func request(…)-> URLSessionTask {
        …
    }
}
```

- Any Class, Struct or Enum can override this implementation

```
extension protocol Networking {
    func request(…)-> URLSessionTask {
        …
    }
}
```

QUESTIONS?

- We'll tackle them now as we code together