A Swift Introduction

Eddie Kaiger

Tuesday, October 20th, 2015

Assumed Knowledge & Tools

- **A Mac with Xcode installed**
- **Some knowledge of programming (ECS 30/40 level)**

Feel free to follow along by opening up a Playground in Xcode (File > New > Playground)

A Hello World program

```
1
2 print("Hello world!")
3
```

Yep, that's it

Swift is a scripting language, which means the language is *interpreted*, not compiled

Also, no semicolons!

Variables

```
1
2 var x: String
3 var y = "Sean Davis"
4
5 var myInt = 4
6 var myDouble: Double = 5
7
```

Swift can (usually) figure out the type of the variable you initialize

Variables can be *implicitly* or *explicitly* typed

implicitly: Swift figures it out

explicitly: We specify the type

Variables

***var* represents a variable that can change**
***let* represents a variable that can NOT* change**

```
1
2 let pi = 3.14
3 pi = 5  // Nope, it's constant
4
5 var gpa = 3.9
6 // Take a few ECS classes...
7 gpa = 2.8
8
```

*Exception: classes declared with *let* can still change ¯_(ツ)_/¯

Arrays and Dictionaries

```
1
2 var numbers = [2, 5, 10]
3 numbers[2]
4
5 var fruit = ["apple": 6, "orange": 10]
6 fruit["apple"]
7
```

```
[2, 5, 10]
10

["apple": 6, "orange": 10]
6
```

Arrays and dictionaries are accessed using bracket notation

Arrays are accessed by the index
Dictionaries are accessed by the key

```
1
2 var values: [String: AnyClass]
3 var arr = [Double]()
```

Variable Modifiers

You can modify how variables are set and returned with *set* and *get* modifiers

```
24 var sideLength: Double = 0
25
26 var perimeter: Double {
27     set {
28         sideLength = newValue / 4.0
29     }
30     get {
31         return sideLength * 4.0
32     }
33 }
34
```

Variable Modifiers

You can also have modifiers such that allow you to get callbacks on whenever a variable *will* be set and a variable *was* set

```
36 var items: [Int] {
37     didSet {
38         tableView.reloadData()
39     }
40 }
```

For Loops

```
3  for i in 1...3 {
4      // Do something with i (includes the 3)
5  }
6
7  for i in 1..<3 {
8      // Does not include 3
9  }
10
```

```
13 var names = ["Alex", "Shyam"]
14 for name in names {
15     print("\(name)")
16 }
```

Conditionals

- **Parentheses *not needed* around condition**
- **Braces *are needed* around statement**

```
5
6  if finalsWeek {
7      drinkCoffee()
8  } else {
9      drinkMoreCoffee()
10 }
11
12 let standing = units >= 39 ? "good" : "bad"
13
```

Enums and Switch Statements

```
enum ShirtSize {
    case Small
    case Medium
    case Large
}

let size = ShirtSize.Large

switch size {
case .Small:
    print("Small shirt")
case .Medium:
    print("Medium shirt")
default: break
}
```

Functions

```swift

func boom() {
    print("yo")
}

func messageForName(name: String, yearOfBirth year: Int) -> String {
    return "\(name) was born in \(year)"
}

print(messageForName("Suzie", yearOfBirth: 1995))
```

Parameters can have a description.

In Swift (and Objective-C), we prefer readability over brevity. Functions should read like a sentence, and variables should be clearly named. Both variables and functions should be in camelCase style.

Functions

```
12 // Alernatively, you can leave out the second description
13 func messageForName(name: String, year: Int) -> String {
14     return "\(name) was born in \(year)"
15 }
16
17 print(messageForName("Suzie", year: 1995))
18
```

Classes and Inheritance

```
 9 class Staff: NSObject {
10     var name: String
11
12     init(name: String) {
13         self.name = name
14         super.init()
15     }
16 }
17
18 class Professor: Staff {
19     var courses = [Int]()
20
21     func addCourse(courseID: Int) {
22         courses.append(courseID)
23     }
24 }
```

Note: non-optional variables must be initialized in *init* before calling *super.init()*

Optionals (now we're talking Swift)

A variable is declared *optional* if the value it contains should be allowed to be *nil* (absent)

Denoted by a question mark:

```
1
2 var middleName: String?
3
```

Optionals (now we're talking Swift)

Accordingly, non-optionals *cannot* be *nil*

```
1
2 var middleName: String? = nil
3 var firstName: String = nil
4
```

Nil cannot initialize specified type 'String'

Optional Chaining

Optional chaining allows us to unwrap objects to make sure they're not nil before accessing items inside of them.

```
71 var student: Student? = Student()
72 let name = student?.courses?[0].professor?.firstName
73
```

The question mark is read as:
"If this item is not nil, then keep going down the chain.
Otherwise, the whole thing is nil and we're done."

```
1
2 var courses: [String]? = ["Data Structures", "Algorithms"]
3 print("My first course is ECS \(courses?.first)")
4 courses?[1].lowercaseString
5
```

```
["Data Structures", "Algorithms"]
"My first course is ECS Optional("Data Structures")\n"
"algorithms"
```

Force Unwrapping

```
21
22 let student: Student? = Student()
23 student!.courses![1].professor?.firstName
24
```

Force unwrapping is when you as the programmer insist that an optional variable is not nil (denoted with an exclamation point). In the case that the variable is nil, you will get a runtime error, so be careful if/when you use it!

If-Let Unwrapping

If-let provides us with a temporarily unwrapped variable to use to avoid force unwrapping or too much optional chaining (for readability)

```
21
22 var student: Student?
23
24 if let s = student {
25     print(s.courses)
26 } else {
27     print("Student is not available.")
28 }
29
```

Protocols

A protocol is a set of functions and variables that other classes can promise to implement

```
1
2  import UIKit
3
4  protocol ShapeDrawing {
5      func drawShape()
6      var color: UIColor { get }
7  }
8
9  class MyShape: NSObject, ShapeDrawing {
10
11     func drawShape() {
12         // Drawing code...
13     }
14
15     var color: UIColor {
16         return UIColor.redColor()
17     }
18 }
19
```

Extensions

Extensions allow you to add new functionality to existing classes

```
3 extension Double {
4     var ft: Double { return self / 3.28084 }
5 }
6
```

Extensions

Extensions allow you to add new functionality to existing classes

```
3 extension Double {
4     var ft: Double { return self / 3.28084 }
5 }
6
```

Extensions

Extensions allow you to add new functionality to existing classes

```
3 extension Double {
4     var ft: Double { return self / 3.28084 }
5 }
6
```