Lecture 2

Swift & the iOS Ecosystem

CIS 195-201 - Dominic Holmes

Attendance + Survey

tiny.cc/cis195-lec2

Today

- **Swift**
  - A LOT to cover: variables, types, arrays, enums, loops, functions, structs, classes
  - These are the basics. More as we go through the semester!
- iOS app structure (might not get to this)
  - Intro to UIKit
  - UIKit vs SwiftUI

Logistics

Spring 2020 19x Lecture Topics

- ~~21 Jan. — Linux/Unix commands~~
- 28 Jan. — Version control with Git + GitHub
- 4 Feb. — HTML/CSS/Internet Basics

- These will be useful! If you don’t know these topics, you should go.

Canvas and Piazza

- Canvas — for submissions. **Always zip your submissions.**
- Piazza — for all Q&A, announcements, tutorials, and apps
  - If you have a private question — Piazza post **>>>** email
  - You'll get a faster response + my sanity preserved
  - Link in the attendance

Waitlist

- **https://forms.cis.upenn.edu/waitlist**
- Course cap is **24** this year
  - If you're not (still) not sure about the class — please decide asap!
- Waitlist placement is based on **attendance**

Swift

What is Swift?

- Apple's open source programming language
  - Created 2014
  - Intended to replace Obj-C for native app development, and C for embedded dev.
- Current version: **5.2**
  - Swift releases can come with *significant* changes (especially 2->3). Google carefully!

“Our goals for Swift are ambitious: we want to make programming simple things easy, and difficult things possible”

– swift.org

```
// This is a comment, by the way. Use /* … */ for multiline.

// Variables store values, and can be changed
var myStr = "Hello, world"
myStr = "New value"

// Constants cannot be changed
let myInt = 42



// Plus lots of other cool features (covered later)
// For now, let's keep it simple
```

Type System

Called “Type Inference”

```
var example = 9
// Assigned the Int type automatically
// We can no longer use ‘example’ for a String, Bool or anything else

// Swift assumes the type when it can, but we can always be more
explicit
let course: String = "CIS-195-201"
let isAmazing: Bool = true
let year: Int = 2020
let percentageFull: Double = 0.90

// Everything in Swift revolves around types
// Types are capitalized (String, Bool, Int, MyClass, MyProtocol...)
```

“Static types can catch bugs at compile time instead of runtime, and directly improve tooling experience like code completion/intellisense, jump to definition, etc.”

“On the other hand, statically typed languages can require a lot of ceremony and boilerplate.”

– Why Swift for Tensorflow?

Strings

```
// Swift is very picky about what types go where

let myNum = 42.0
let myFruit = “straw” + “berries”
print(myNum)                          // This works
print(myNum + " " + myFruit)          // This does NOT

// We can't add a Double to String, we must convert
print(String(myNum) + " " + myFruit)

// We can also use "string interpolation" by adding a \() in the
middle of a string
print("I have \(myNum) oranges”)
```

Collection Types

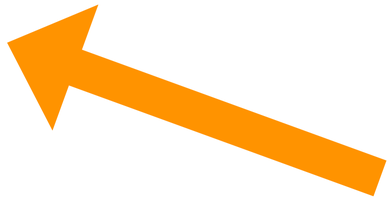
```swift
let heartRate: [Double] = [50.0, 55.3, 78.2]
let workoutTypes: [String] = ["Run", "Swim", "Walk"]

// Swift can also assume these types
let activeMinutes = [221, 42, 103, 49, 50]

// Things can get wild
let workoutsToMinutes: Dictionary<String, [Int]> = [
    "Run" : [54, 14, 16],
    "Walk" : [41, 22, 19]
]

// Type inference is good at what it does
let workoutsToHeartRate = [
    "Run" : [50.0, 55.3, 78.2],
    "Walk" : [50.0, 55.3, 78.2]
]
```

What's the type?



What's the type?

Enums

```
// A way to define different cases a value could take
// For example: if we have a bounded set of workout types:
enum Workout {
    case run
    case swim
    case walk
}

let myWorkoutToday = Workout.run
var myWorkoutTomorrow: Workout = .swim

myWorkoutTomorrow = .walk // changed my mind

let workoutsToFrequency: Dictionary<Workout, Int> = [
    .run: 30, .walk: 45, .swim: 19
]
```

Loops

```
let crowd = ["percy", "romulus", "harry", "aberforth"]
for person in crowd {
    print("Hey \(person)!")
}

// Could use a Range to do the same thing
for personIndex in 0 ..< crowd.count {
    print("Hey \(crowd[personIndex])")
}

var harryAlive = true
while harryAlive {
    print("A horcrux remains!")
    harryAlive = false
}
```

Functions

```
func getHelp() {
    print("Try posting on piazza 🤖")
}
getHelp() // Calls the function

func begin(workout: Workout, minutes: Int) {
    if workout == .swim {
        print("Swim for \(minutes)!")
    }
}
begin(workout: .swim, minutes: 90)

func verify(_ workout: Workout, for minutes: Int) -> Bool {
    return (workout == .swim && minutes > 60)
}
let goodWorkout = verify(.swim, for: 90)
```

Functions

```
func NAME(OUTSIDE INSIDE: TYPE, _ INSIDE: TYPE) -> RETURNTYPE {
    . . .
}
```

```
func NAME(INSIDE-AND-OUTSIDE: TYPE) -> RETURNTYPE {
    // omitting the outside label entirely just uses the inside
}
```

Structs and Classes

```
// Two ways to define our own "objects"
// Struct is value type, Class is reference type (like Java's classes)

struct Car {
    var paint: UIColor
    let model: String
}

let myCar = Car(paint: .blue, model: "Honda")


class Bike {
    var paint: UIColor
    let model: String

    init(paint: UIColor, _ model: String) {
        self.paint = paint
        self.model = model
    }
}

let myBike = Bike(paint: .yellow, "Ducati")
```

```
var myCarCopy = myCar
myCarCopy.paint = .white

dump(myCar)      // blue
dump(myCarCopy)  // white
```

```
let myBikeCopy = myBike
//how can we use let?
myBikeCopy.paint = .white

dump(myBike)     // white
dump(myBikeCopy) // white
```

pass by reference

Use When….

- Comparing identity by *reference* makes sense
- You want shared, mutable state

pass by value

Use By Default. Especially when…

- Comparing identity by *actual value* makes sense
- You want copies to have independent state
- *You want more efficient, testable, predictable code*

Optionals

```
// Optionals are a way of dealing with the possibility that a value is NIL
var newStudent: String? = nil

let nameLength = newStudent?.count
// nameLength is of type "Int?". There could be a name length (Int), or nothing (nil)
// There are a few ways of dealing with optionals to get the real values

// Unwrapping
if let unwrapped = newStudent {
    print(unwrapped.count) // unwrapped only defined inside this context, and is of type "Int"
} else {
    print("New Student is nil")
}

// Nil-coalescing (fancy name for: use a default value if nil!)
let unwrapped = newStudent ?? "Default Value"

// Force unwrapping
let unwrapped = newStudent!
// This will crash your app if newStudent is nil. ALWAYS AVOID using this.
```

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965… This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

– Sir Charles Anthony (Tony) Hoare

Swift is great because...

- It's **safe** (optionals + types)
- It's **fast**
- It's **expressive — easy to read and write**
- *Simple* to write easy things (hello world, iOS apps)
- *Possible* to write hard things (an OS, a DSL, etc)

Live Demo: Mosaic

Due Before Next Class

- **App 1: Swift**
- **Tutorial 1: iOS Architecture**

Links

- Survey: **tiny.cc/cis195-lec2**
- Piazza: **tiny.cc/cis195-piazza**