Announcements

- **Next Wednesday (Sept 4th) is our first "Studio Day"**
  - If you have a Mac laptop running Xcode 10.2.1 come to Steinberg 105 during the normal class time (11:30 – 1 PM)
  - If you do not have a Mac laptop you will meet in Whitaker 316 either during class time or on Thursday
    - I will email out your Whitaker lab time based on the Google doc survey responses
- **If you have a Mac laptop**
  - Install Xcode 10.2.1 before class
    - https://developer.apple.com/download/more/

Today's Topics

- **Swift**
  - Overview
  - Syntax
  - Examples
- **Xcode 10**
  - Playgrounds

Swift

- New programming language developed by Apple
- Announced at WWDC 2014
- Interoperates with Objective-C
  - Both are considered first class citizens
- We are using Swift version 5

Hello World in Swift

```
print("Hello World")
```

- No semicolons
- No main method needed

Variables and Constants

- **Swift uses var and let to describe variables and constants**
- **Variables and constants have a type**
  - let languageName: String = "Swift"
  - var version: Double = 1.0
  - let isEverChanging: Bool = true
- **Swift supports type inference**
  - let languageName = "Swift " //inferred as String
  - var version = 1.0 //inferred as Double
  - let isEverChanging = true //inferred as Bool

Common Data Types in Swift

- **String**
- **Character**
- **Int**
- **Float**
- **Double**
- **Bool**
- **Optional**

Strings

- **Swift makes working with strings easy**

```
let firstName = "John"
let lastName = "Smith"
let fullName = firstName + " " + lastName
```

- **Enumerating through them is familiar**

```
for character in firstName.characters{
    print(character)
}
```

J
o
h
n

String Interpolation

```
let a = 2, b = 3

// "2 times 3 is 6"
let mathResult = "\(a) times \(b) is \(a * b)"
```

Collections - Arrays and Dictionaries

var names = ["Bob", "Alice", "Mike", "Jen"]

– Inferred as a typed collection of Strings

- **I could also be more explicit:**
  var names: [String] = ["Bob", "Alice", "Mike", "Jen"]

var numberOfLegs = ["ant": 6, "snake": 0, "cow" :4]

– Inferred as a typed dictionary of Strings and Ints

- **Or I could be more explicit:**
  var numberOfLegs: [String: Int] = ["ant": 6, "snake": 0, "cow" :4]

Collections – Sets

A collection that stores distinct elements with no defined order

```
var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]
```

```
var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
```

– Inferred as a set of type Set<String> collection of Strings

```
print("I have \(favoriteGenres.count) music generes.")
 //Prints "I have 3 favorite music genres."

 if favoriteGenres.isEmpty {
   print("Nothing here")
}
```

- **Add unique strings to the set**
  favoriteGenres.insert("Jazz")

Loops

```
while !done {
  keepDoingSomething()
}
for num in 1...5 {    //Prints from 1 up to and including 5
  print("\(num) times 4 is \(num * 4)")
}

for num in 1..<5 {    //Prints from 1 up to 4
   doSomething(i)
}
```


Conditionals

```
if legCount == 0 {
   print("Does not walk")
} else if legCount == 1 {
  print("Hopping around")
} else {
   print("I can walk")
}
```

```
switch legCount {
  case 0:
   print("Does not walk")

 case 1, 3, 5, 7:
   print("Limps around")

default:
   print("I can walk")
}
```

Functions

```
func sayHi() {
   print("Hi")
}
    sayHi()

func sayHi(name: String) {
   print("Hi \(name)!")
}
    sayHi(name: "Bob")
```

```
func sayHi(name: String = "CSE 438") {
   print("Hi \(name)!")
 }
    sayHi()            //Prints Hi CSE 438
    sayHi(name: "Bob")     //Prints Hi Bob
```


Functions

```
func sayHi(name: String = "CSE 438")  -> String {
   return "Hi " + name
 }
    let name =  sayHi()      //Name contains "Hi CSE 438"
```

```
func refreshWebSite() -> (Int, String ) {
   // refresh
   return (200, "Success")
 }
    let (statusCode, message) =refreshWebSite()
```

Closures

- Self-contained blocks of functionality that can be passed around

```
let displayGreeting = {
   print("Hello Class")
}

let displayGreeting: () -> () = {
   print("Hello Class")
}

displayGreeting()
```

//Inferred as this

//looks very similar to a function (named closure)

Optionals

- Optionals handle the absence of a value
  - There is a value and it equals x or there isn't a value

```
var numberOfLegs = ["ant": 6, "snake": 0, "cow" :4]
let possibleNumLegs  = numberOfLegs["goat"]   ???
let possibleNumLegs: Int?  = numberOfLegs["goat"]  //Value or nil

If possibleNumLegs != nil {
    let legCount = possibleNumLegs!    //Use ! to unwrap the optional
   print("Goat has \(legCount) legs")
}
```

- Shorthand for above, if let

```
If let legCount = possibleNumLegs {
   print("Goat has \(legCount) legs")
}
```

Enumerations

- A common type for a group of related values
- Much more powerful than enumerations in the C language
- Allows for associated values of ANY type (not just integer values)

```
enum CompassPoint {
  case north
  case south
  case east
  case west
}
```


Enumerations

```
enum CompassPoint {
  case north
  case south
  case east
  case west
}
var directionToHead = CompassPoint.west
directionToHead = .south

  switch directionToHead {
  case .north:
    print("Lots of planets have a north")
  case .south:
    print("Watch out for penguins")
  case .east:
    print("Where the sun rises")
  case .west:
    print("Where the skies are blue")
  }
 // Prints "Watch out for penguins"
```

Classes and Structures (structs)

- General purpose constructs which are the building blocks of your code
- You define **methods** and **properties** to add functionality

- Classes have additional capabilities that structs do not
  - Inheritance enables one class to inherit characteristics of another
  - Type casting allows you to treat an instance as a superclass or subclass from their class hierarchy

Classes

```
class Person {
  var age = 21    //defines the properties

  var description: String {   //defines a computed property
      get {
        return "You are \(age) years old"
      }
   }
}

let somePerson = Person()
print("Hello, you are \(somePerson.age) years old")
```

Properties

- **Associated values with a particular class, struct, or enum**
- **Properties are either stored or computed**
  - Stored properties are constants and variables associated with an instance
    - Not available in an enum
  - Computed properties are calculated

```
struct FixedLengthRange {
   var firstValue: Int
   let length: Int
}

var rangeOfThreeItems = FixedLengthRange(firstValue: 0, length: 3)
 // the range represents integer values 0, 1, and 2

rangeOfThreeItems.firstValue = 6
 // the range now represents integer values 6, 7, and 8
```


Extensions

- **Adds new functionality to an existing structure, class, enumeration or protocol**
- **Extensions support the following features:**
  - Add computed instance and type properties
  - Specify instance and type methods
  - Make existing type conform to a protocol
- **Extensions may add new functionality to a type, but are unable to override existing functionality**

Extensions

```
extension Double {
  var km: Double { return self * 1_000.0 }
  var m: Double { return self }
  var cm: Double { return self / 100.0 }
  var mm: Double { return self / 1_000.0 }
  var ft: Double { return self / 3.28084 }
}

  let oneInch = 25.4.mm
  print("One inch is \(oneInch) meters")
// Prints "One inch is 0.0254 meters"

  let threeFeet = 3.ft
  print("Three feet is \(threeFeet) meters")
// Prints "Three feet is 0.914399970739201 meters"
```


More Information about Swift Language

- **Official Swift Programming Guide**
  - https://docs.swift.org/swift-book/index.html
- **WWDC 2016 – 2019 Videos**
  - developer.apple.com

Examples in Playground