ATLS 4120/5120: Mobile Application Development
Week 4: Swift Intermediate

Go into Xcode
File | New | Playground
A playground is a new type of file that allows you to test out Swift code, and see the results of each line in the sidebar.
Name: swift2
Platform: iOS
Save
Delete what's there so you start with an empty file

Classes

- A class provides a template, or blueprint for its objects.
- A class defines the characteristics (data properties) and behavior (methods) of its objects.
  - Properties associate values with a class
  - Methods are functions that are associated with a class
- Classes should have UpperCamelCase names to match the capitalization of standard Swift types.
- Classes are reference types so they are passed by reference, not copied when they are assigned to a variable/constant, or passed to a function.
- Structs and enumerations are value types, so they are copied when assigned or passed.

```
class Vehicle {
    var wheelNum = 4
    var speed = 25
    var mpg = 20
    let tankCapacity = 20
    var name : String?
    func changeSpeed(amount: Int){
        speed = speed + amount
    }
    func changeEfficiency(speed newSpeed: Int, mpg newmpg: Int){
        speed = newSpeed
        mpg = newmpg
    }
}
```

Objects

- An object is an instance, or occurrence, of a given class.
  - An object of a given class has the structure and behavior defined by the class
  - Many different objects can be defined for a given class
  - All objects of the same class have the same structure
- An instance of a class is traditionally known as an object. However, Swift classes and structures are much closer in functionality than in other languages, and a lot of the functionality can apply to instances of either a class or a structure type. Because of this, the more general term instance is used.
- Create an instance of a class by calling an initializer method.
- The simplest initializer syntax is () which calls the default init()
- Initializers make sure that every stored property has a value when the initialization completes
- Properties are accessed using dot notation

```
let myJeep = Vehicle()
myJeep.mpg
myJeep.speed
```

Methods

- Instance methods are functions that belong to a class
  - Same syntax as functions
  - Call methods with the same dot notation as properties
- Like functions, method parameters can have a both a local name and external name.
- In methods by default the first parameter name is local and subsequent parameter names are automatically both local and external
  - The goal is to make method calls clear
- Method headers in the documentation will show the first parameter as _: since it doesn't have an external name.

```
myJeep.changeSpeed(10)
myJeep.speed
myJeep.changeEfficiency(speed: 35, mpg: 25)
myJeep.speed
myJeep.mpg
```

Swift also has type methods

- functions that are called on the class itself
- Use the keyword 'static' before the function

Initialization

- Initialization is the process of preparing an instance of a class for use
- A class includes methods used to create and initialize a new instance of a class called initializers.
  - They ensure that the new instance is correctly initialized before they are used the first time
- During initialization an initial value must be set for each stored property on that instance
  - Default value
  - Initial value
- Swift provides a default initializer called init() for a class that has default values for all its properties
- You can also create your own **init()** methods to provide initial values that don't have defaults
- Because initializers don't have descriptive names(init), Swift provides an automatic external name for every parameter in an initializer if you don't provide an external name yourself.
- Because initializers don't have descriptive names(init), if you don't provide external parameter names Swift will use your local parameter names as external parameter names as well.
- If you call an initializer method without a parameter name you will get a compile error
- If you don't want an external parameter name use an underscore _

[Update Vehicle class]

```
    init(vehicleName vname: String){
        name = vname
    }
```

But now you get errors where you defined myJeep because there's no empty init() method. So you must add one to the class.

```
    init(){
    }
```

```
let myHybrid = Vehicle(vehicleName: "Prius")
myHybrid.name
```

It says {Some "Prius"} because name is an optional.
You need to check that it's not nil and then force unwrap it

```
if myHybrid.name != nil {
   println(myHybrid.name!)
}
```

Inheritance

- Inheritance enables classes to form a class hierarchy like a family tree.
- Allows subclasses to share the structure and behavior of its superclass.
  - Superclass is the parent class
  - A subclass extends a class
    - Inherits from the superclass
    - Can add properties and methods
    - Can modify inherited properties
- A subclass can provide its own custom implementation of methods or properties through overriding
  - Prefix your overriding definition with the keyword 'override'
  - Overriding says that you intend to provide an override, not that you're providing a matching definition by accident
- When creating an initializer in a subclass, set your own properties and then you must call the superclass's initializer **super.init()**

```
class Bicycle : Vehicle {
    var reflectors = true
}
```

```
var bike=Bicycle()
bike.wheelNum
bike.wheelNum = 2
bike.wheelNum
```

`bike.reflectors` is true
`myJeep.reflectors` error – class Vehicle doesn't have a reflectors property

change Bicycle

```
var reflectors : Bool
```

Get errors because reflectors doesn't have a value.

```
    init(_ ref : Bool){
        reflectors=ref
        super.init()
    }
```

When creating an initializer in a subclass, set your own properties and then you must call the superclass's initializer super.init()
Super.init() calls Vehicle's init()

```
var bike=Bicycle(false)
```

Collection Types

- Swift has three types of collections
  - Arrays
    - ordered collections of values
  - Sets
    - unordered collections of distinct values
  - Dictionaries
    - unordered collections of key/value pairs
- The collection will be mutable if it's assigned to a variable, immutable if it's assigned to a constant
- Properties
  - .count returns the number of items in an array
  - .isEmpty is a boolean to see if count is 0

Arrays
Arrays store an ordered collection of values
Arrays start with an index of 0 just as in other languages

- removeAtIndex and removeLast return the removed item
- insert(_:atIndex:) inserts an item into the array at a specified index

```
var shoppingList=["cereal", "milk"]
print(shoppingList[0])
shoppingList.append("bread")

if shoppingList.isEmpty{
    print("there's nothing you need")
} else {
    print("You need \(shoppingList.count)" + " items")
}

let item = shoppingList.removeLast()

for item in shoppingList{
        print(item)
}
```

Dictionaries
Dictionaries store unordered key/value data pairs
.keys returns all the keys
.values returns all the values
updateValue() returns the old value for that key
removeValueForKey() returns the removed value or nil if no value existed

```
var newList=[String:String]()
```

```
var classes:[String: String]=["4120":"MAD", "3000":"Code"]
```

As with arrays, you don't have to write the type of the dictionary if you're initializing it with a dictionary literal whose keys and values have consistent types.

```
classes["3000"]
classes["2000"]="MIT"
classes.count

classes.updateValue("Mobile App Dev", forKey: "4120")
classes.removeValueForKey("3000")

for (num, name) in classes{
    print("\(num): \(name)")
}
```

Memory Management

- Swift uses Automatic Reference Counting (ARC) to manage memory usage
- ARC automatically frees up the memory used by class instances when those instances are no longer needed
- Automatic Reference Counting (ARC) was introduced in iOS5 so you don't have to worry about it. Yeah!