GENERAL ASSEMBLY

MOBILE DEVELOPMENT
DEEPER INTO SWIFT

Kishin Manglani

AGENDA

- Recap
- Collection Types
  - Arrays
  - Sets
  - Dictionaries
- For Loops
- Inheritance
- Protocols

RECAP

INITIALIZERS

- Initializers are called to create a new instance of a particular type. In its simplest form, an initializer is like an instance method with no parameters, written using the init keyword

```
init() {
    // perform some initialization here
}
```

REMEMBER UIKIT?

- We can reuse all of Apple's code
- So what do I mean reuse? We can inherit from the UIViewController class
- What does inherit mean? We get all the "parent class's" variables and functions
- We add this colon and the other class we want to inherit from

```
class MyViewController: UIViewController {



}
```

REMEMBER UIKIT?

- Now the MyViewController class has all of the variables and all of the functions that UIViewController has
- MyViewController (child class) is a subclass of UIViewController (parent)
- UIViewController (parent) is the superclass of MyViewController (child)
- Since it is a UIViewController, Apple and Storyboard know how to draw it and know it has a view, etc.

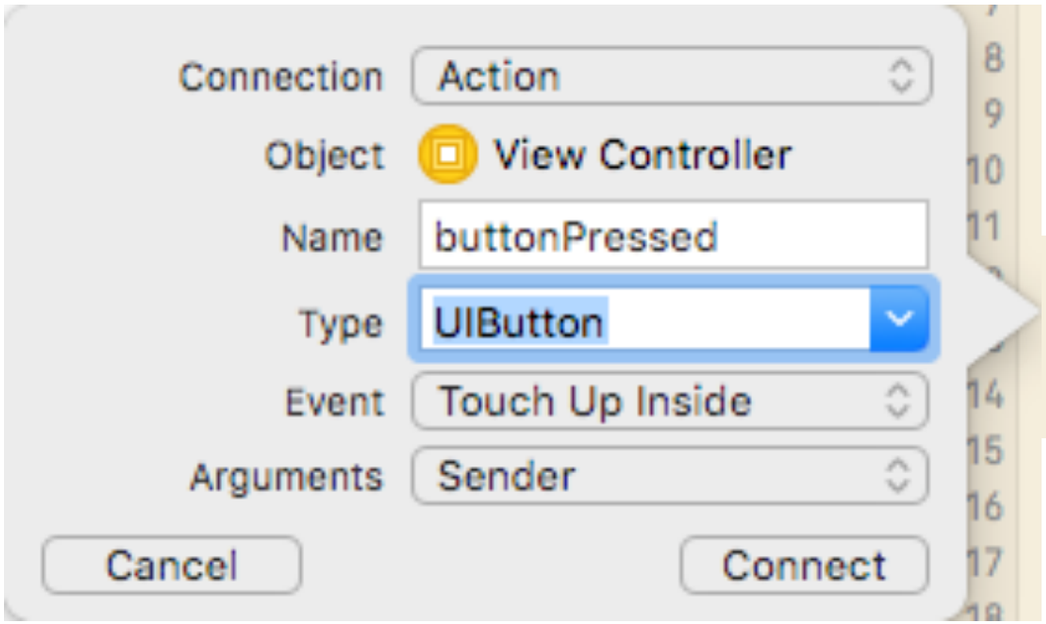
```
class MyViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        print("hello world. this is the console")
    }

}
```

IBOUTLET

- IBOutlet or Interface Builder Outlet
- Create by control + dragging from our storyboard to our view controller file in the assistant editor
- What is this doing? This is creating a variable in our code that we can reference later
- So we are adding this variable to our subclass of UIViewControllers

```
@IBOutlet weak var nameLabel: UILabel!
```

IBACTION




```
@IBAction func buttonPressed(sender: UIButton) {

}
```

OBJECTIVES

- More on inheritance: override and super
- Learn how to use Arrays
- Learn how to use For Loops
- Learn how to use Github to submit homework

COLLECTION TYPES

ARRAYS

ARRAY

- Let's think about properties in our classes again
- In our city example, how would we store multiple buildings in the city class?

```
var shoppingList = ["Eggs", "Milk"]
```

```
var shoppingList: [String] = ["Eggs", "Milk"]
```

ARRAY

- An array is a collection data type
- An array stores values of the same type in an ordered list
- The same value can appear in an array multiple times at different positions
- Remember, this is ordered!

```
var shoppingList = ["Eggs", "Milk", "Eggs"]
```

ARRAY

- How do we access elements in an array?
- Arrays are zero-based, in other words the first element is really the zeroth element
- Subscript and index

```
var shoppingList = ["Eggs", "Milk"]
shoppingList[0]
shoppingList[1]
```

```
["Eggs", "Milk"]
"Eggs"
"Milk"
```

ARRAY

- Arrays have some useful properties
  - count: returns the number (Int) of objects in the array
  - isEmpty: returns a Bool checking if the array is empty (0 objects)
- Append method: adds an object to the array
  - shoppingList.append(“carrots”)

```
var shoppingList = ["Eggs", "Milk"]
shoppingList.append("Carrots")
shoppingList[2]
shoppingList.isEmpty
shoppingList.count
```

```
["Eggs", "Milk"]
["Eggs", "Milk", "Carrots"]
"Carrots"
false
3
```

ARRAY

- Creating an empty array

```
var someInts = [Int]()
```

ARRAYS

- Methods
  - insert(value, atIndex: index)
  - removeAtIndex(index)
  - removeLast()

- Properties:
  - count: returns an Int
  - isEmpty: returns a Bool

SETS

SET

- Stores unordered values of the same type
- Similar to an array, but is unordered
- Better performance for contains and other operations than arrays, but performance is not important at the moment
- Don’t worry about sets too much, focus on arrays

```
var shoppingSet: Set = ["Eggs", "Milk"]
shoppingSet.contains("Eggs")
```

DICTIONARIES

DICTIONARIES

- Associates keys of the same type with values of the same type
- No defined/specified ordering
- Each value is associated with a unique key, which acts as an identifier for that value within the dictionary
- Similar to a traditional dictionary: we use a key (word) to look up a value (definition)
- Provides more meaning than merely index numbers

```
var favoriteColors = ["Kishin" : "blue", "John" : "green"]
favoriteColors["Kishin"]
favoriteColors["Kishin"] = "orange"
favoriteColors["Kishin"]
```

DICTIONARIES

- Creating an empty dictionary:

```
var namesOfIntegers = [Int: String]()
// namesOfIntegers is an empty [Int: String] dictionary
```

DICTIONARIES

- Adding values:

```
namesOfIntegers[16] = "sixteen"
// namesOfIntegers now contains 1 key-value pair
namesOfIntegers = [:]
// namesOfIntegers is once again an empty dictionary of type [Int: String]
```

DICTIONARIES

- Creating a dictionary from a literal:

```
var airports = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
```

DICTIONARIES

- Check if a value exists:

```
if let airportName = airports["DUB"] {
    print("The name of the airport is \(airportName).")
} else {
    print("That airport is not in the airports dictionary.")
}
```

DICTIONARIES

- Methods
  - updateValue(value, key)
  - removeValueForKey(key)

- Properties:
  - count: returns an Int
  - isEmpty: returns a Bool
  - keys: returns an array of all keys in the dictionary
  - values: returns an array all values in the dictionary

FOR LOOPS

FOR LOOPS

- So arrays allow us to store an undetermined amount of data
- How can we process or perform some logic on an undetermined amount of data?
- For loops
- Begin and end with curly braces
- Run code between curly braces 0 or more times

FOR LOOPS

```
for var i = 0; i < 3; i++
{
    print(i)
}
```

FOR LOOPS

```
for i in 0...2 {
    print(i)
}
```

FOR LOOPS

```
var shoppingList = ["Eggs", "Milk", "Cheese"]
for item in shoppingList {
    print(item)
}
```

```
["Eggs", "Milk", "Cheese"]

(3 times)
```

PLAYGROUND

INTRO TO SWIFT

INHERITANCE

INHERITANCE

```
class Hat {
    var color: String
    var size: Int

    init(newColor: String, newSize: Int) {
        self.color = newColor
        self.size = newSize
    }
}
```

INHERITANCE

- A class can inherit methods, properties, and other characteristics from another class. When one class inherits from another, the inheriting class is known as a subclass, and the class it inherits from is known as its superclass

```
class SomeSubclass: SomeSuperclass {
    // subclass definition goes here
}
```

INHERITANCE

- A subclass can provide its own custom implementation of something that it would otherwise inherit from a superclass. This is known as overriding.
- Do this using the override keyword
- Why? Clarifies intent to provide an override and have not provided a matching definition by mistake

DEMO

INTRO TO SWIFT

PROTOCOLS

PROTOCOLS

- What is the standard definition of a protocol?
- In the real world, people on official business are often required to follow strict procedures when dealing with certain situations. Law enforcement officials, for example, are required to “follow protocol” when making enquiries or collecting evidence. -Apple

PROTOCOLS

- A group of related properties and methods that can be implemented by any class
- More flexible than a normal class interface, since they let you reuse a single API declaration in completely unrelated classes
- Also, we don’t have to override or implement a method from a superclass
- With a protocol we can make certain methods required and others optional

PROTOCOLS

- How do we define a protocol?

```
protocol SomeProtocol {
    // protocol definition goes here
}
```

PROTOCOLS

- How do we have a class adopt a protocol?

```
class SomeClass: SomeSuperclass, FirstProtocol, AnotherProtocol {
    // class definition goes here
}
```

PROTOCOLS

- Here's an example:

```
protocol Swimmer {
    func swim()
}
```

- Notice how we do not need to provide an implementation (any code for the function) it is just the definition

PROTOCOLS

- Here's an example:

```
protocol Swimmer {
    func swim()
}
```

- Notice how we do not need to provide an implementation (any code for the function) it is just the definition

PROTOCOLS

- Here's how we adopt the protocol

```
class Frog: Animal, Swimmer {
    func swim() {
        print("I'm swimming")
    }
}
```

- When adopting the protocol we need to implement the required methods