iOS Foundations II
Day 8

- Homework review
- For loops
- Data persistence
- What's next?

Control flow and `for` Loops

- Swift provides **control flow** statements whose syntax is similar to other C-like languages.
- Swift provides these control flow statements:

  `for`, `for-in`, `while`, `do-while`, `if`, `if-else`, `switch`

- We'll focus on `for` loops today, covering the two main ways that Swift lets you implement them.

`for` Loops

- Swift provides two categories of **`for`** loops:
  - “Normal” **`for`** loop: Performs a set (a block) of statements until a specific condition is met. Typically the loop increments a counter each time the loop’s statement block is done executing, then the counter is compared to a threshold value to determine if the loop should continue running.
  - **`for-in`** loop: Performs a set (block) of statements for each item in a range, sequence, collection, or progression.

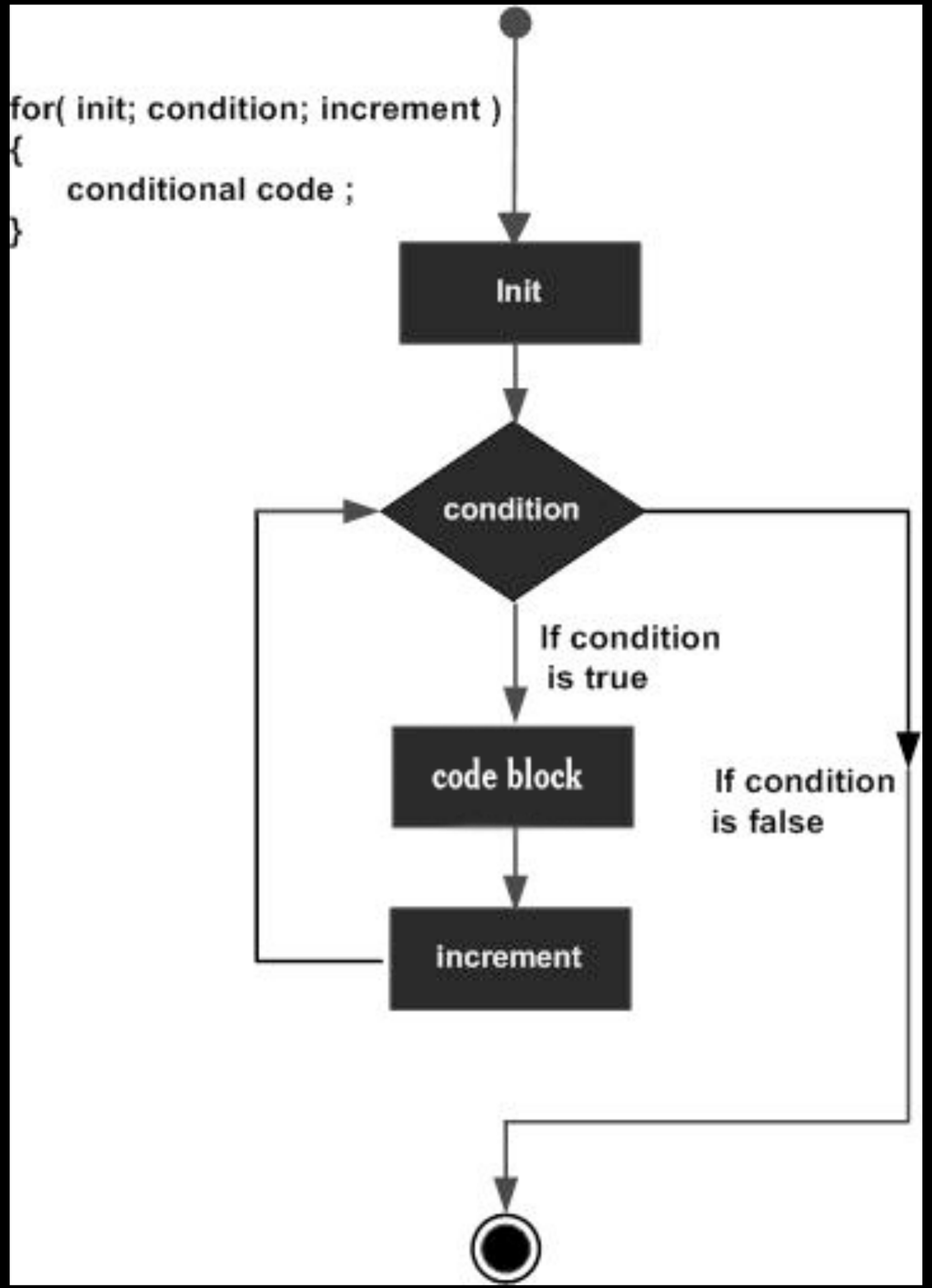
“Normal” for Loop

- General form

```
for initialization; condition; increment {
    statements
}
```

- Example

```
for var index = 0; index < 3; ++index {
    println("index is \(index)")
}
// index is 0
// index is 1
// index is 2
```



`For-in` Loop demo

- Use a **`for-in`** loop to iterate over collection of items.

- **—> See demo! <—**

Demo 8.1:
`for` and **`for-in`** loops

Data Persistence

- Many apps require state (data) to persist. State is preserved between app execution sessions. Typically, state is **serialized** (see slides later in this lecture) & stored locally in a device's flash memory. Data can also be stored on a remote server and accessed by an app over a network.
- Persistence lets a user "continue where he/she left off" after an app has stopped and then restarted.
- We introduce 3 ways to persist data: CoreData, NSKeyedArchiver, NSUserDefaults. We'll go into detail on the latter two ways.

Data Persistence

	Core Data	NSKeyedArchiver	NSUserDefaults
Persists State	**Yes**	**Yes**	**Yes**
Pain in the Ass	**Yes**	**No**	**No**
Speed	**Fast**	**Slow**	**Slow**

Source: NSHipster

Data Persistence

	Core Data	NSKeyedArchiver	NSUserDefaults
Entity Modeling	Yes	No	No
Querying	Yes	No	No
Serialization Format	SQLite, XML, or NSData	NSData	Binary Plist
Migrations	Automatic	Manual	Manual
Undo Manager	Automatic	Manual	Manual

Source: NSHipster

Serialization

- What is **serialization**?
- *Hint*: It's different from "cereal - ization" (har har).

Serialization

- In a typical filesystem, bits are stored in a linear order, one bit after another, one byte after another. The bytes are in **serial** order.
- But data, especially data organized in a 2-D or higher dimensional space (e.g., multispectral image), may not have an “obvious” byte order.(e.g., which pixel to save first? Which color band of a pixel to save first?)

Serialization

- Suppose an app lets a user create an image, and the image has metadata associated with it. How do we save such data?

```
3  struct GPScoords {
4      var latitude:  Float
5      var longitude: Float
6  }
7
8  enum ExposureType: Int {
9      case linear = 1
10     case log
11 }
12
13 struct ImageMetadata {
14     var imageName:  String
15     var location:   GPScoords
16     var exposure:   ExposureType
17     var gamma:      Float
18 }
19
20 var loc = GPScoords(latitude: 122.0, longitude: 45.0)
21
22 var I = ImageMetadata(imageName: "PDX_vacation_001.jpg", location:
       loc,
23                        exposure: ExposureType.linear, gamma: 2.4)
24 I.location.longitude
```

{latitude 122, longitude 45}

{imageName "PDX_vacation_001.jpg", {latitude.

45

Serialization

- Any system that provides data persistence in general* must be able to take any “blob” of data an app hands it and **serialize** it, i.e., the system must write each byte, one after another, until your entire blob is saved.
- The system must also be able to re-create your “blob” by loading the serialized data from persistent storage into memory, and into a format that your app can work with, such as data structures, objects, etc.

*In this context, “General” means the system supports persistence for a variety of data types and combinations of data, such a heterogenous data sets.

NSUserDefaults

- “NSUserDefaults allows an app to customize its behavior based on user preferences”
- Think of it as an automatically persisting plist that is easily modified in code.
- Use the standardUserDefaults class method to return the shared defaults object.
- Setting values inside of it is as easy as these methods:
  - setBool:ForKey:
  - setObject:ForKey:
  - setInteger:ForKey:

NSUserDefaults

- Each app has its own database of user preferences
- Used to store and retrieve an object
- Objects must be NSCoding-compliant
- Primitives may be stored as-is (Float, Int, Bool, String, etc.)
- Try to follow Apple's recommendation of only saving small 'settings' related data in the user defaults.

NSUserDefaults Workflow

- **To save to it:**

1. Get a reference to the standard user defaults singleton
2. set a value with key on the user defaults
3. tell it to ‘synchronize’ aka save

- **To read from it:**

1. Get a reference to the standard user defaults singleton
2. Try to get a reference to a value stored inside of it with a specific key (just like a dictionary)
3. See if the reference you got back contains something, if it does, then that means there was a value there already.

NSUserDefaults examples

To save data:

```
let userInfo = ["Name" : "Brad", "token" : "jfh1234"]

NSUserDefaults.standardUserDefaults().setObject(userInfo, forKey: "userInfo")
NSUserDefaults.standardUserDefaults().synchronize()
```

To load data

```
if let userInfo = NSUserDefaults.standardUserDefaults().objectForKey("userInfo") as?
[String : String] {
    println(userInfo["Name"]) // prints Brad
}
```

Demo 8.2: NSUserDefaults

NSKeyedArchiver

- NSKeyedArchiver/Unarchiver **serializes** NSCoding-compliant classes to and from a data representation.
- **Classes you want to serialize with NSKeyedArchiver must conform to the NSCoding protocol.**
- Once you have done that, it is as simple as calling archive and unarchive on NSKeyedArchiver/Unarchiver to load your object graph
- The amazing part of NSKeyedArchiver is that your object graph is saved and loaded as your custom model types. You don't have to recreate all of your model objects when you load them, like we had to do with the plist.

NSCoding Protocol

- The NSCoding protocol is a very simple protocol, it only has 2 methods:
  - init(Coder)
  - encodeWithCoder()
- Your class that conforms to NSCoding must also inherit from NSObject. See the sample code on the git repo.
- The implementation of these two methods is very much just boilerplate code, as you will see in the next slide.

NSCoding Protocol

```
//first required method is the init with coder, this is used internally by
    NSKeyedUnarchiver to load your objects from the archived data
required init(coder aDecoder: NSCoder) {
    self.firstName = aDecoder.decodeObjectForKey("firstName") as String
    self.lastName = aDecoder.decodeObjectForKey("lastName") as String
    if let decodedImage = aDecoder.decodeObjectForKey("image") as? UIImage {
        self.image   = decodedImage
    }
}

//the other required method, used to encode your objects into the archive file by
    NSKeyedArchiver
func encodeWithCoder(aCoder: NSCoder) {
    aCoder.encodeObject(self.firstName, forKey: "firstName")
    aCoder.encodeObject(self.lastName, forKey: "lastName")
    if self.image != nil {
        aCoder.encodeObject(self.image!, forKey: "image")
    }
}
```

Saving/Loading from disk

- When you use NSKeyedArchiver and NSKeyedUnarchiver, you are saving an archive file to disk. To do this, you will need a path that you want to save to.
- Each app has a separate ‘sandbox’ (or directory) for storing data. iOS keeps the apps separated from each other and the OS for security reasons.
- The sandbox contains a number of different sub directories, and the one we are allowed to write to is called the documents directory.

Getting the path

- We can use the function ‘NSSearchPathForDirectoriesInDomains() to get a path to the documents directory.
- It takes 3 parameters:

1. An enum for which directory you are looking for (we will use .DocumentsDirectory since we want the documents directory!)
2. The domain mask (will always use .UserDomainMask)
3. expandTilde Boolean (will always use true)

Example of a save

```
func saveToArchive() {
    //get path to documents directory
    let documentsPath = NSSearchPathForDirectoriesInDomains(.DocumentDirectory, .
        UserDomainMask, true)[0] as String
    //archive
    NSKeyedArchiver.archiveRootObject(self.people, toFile: documentsPath + "/
        archive")
}
```

Example of a load

```
func loadFromArchive() {

    //get path to your app's documents directory in its sandbox
    let documentsPath = NSSearchPathForDirectoriesInDomains(.DocumentDirectory, .
        UserDomainMask, true)[0] as String
    //attempt to unarchive your object graph
    if let peopleFromArchive = NSKeyedUnarchiver.unarchiveObjectWithFile
        (documentsPath + "/archive") as? [Person] {
        //stored the data we just unarchived into this proper   ty
        self.people = peopleFromArchive
        //this is great, it loaded our stuff
    }
}
```

Demo 8.3: NSCoder and NSKeyed[Un]Archiver

Class Wrap up

- In our 8 sessions together, we covered a ton of ground.
- The big, super important concepts are:
  - MVC*
  - Storyboards
  - Classes & Objects*
  - Methods & Parameters/Return values*
  - ViewControllers (& life cycle methods!)
  - Segues (passing data between them)

*applies to many different programming languages!

What next?

- Keep coding. If you stop, your coding dreams could fade… maybe.
- If you enjoyed learning about iOS and this course, consider:
  - Taking Code Fellows' Development Accelerator (apply now!)
  - Build your own app and submit it to Apple's App Store
- Consider that some of the apps published in the app store are simpler than the app we created in this course.
- Try another programming language. Once you have the basic concepts of objects, methods, variables, parameters, etc — learning a different programming language is considerably less difficult.