**Stanford Course: Developing iOS Apps with Swift**

**Lecture 1: *Intro to iOS 11, XCode 9, Swift 4***

* What’s in iOS: 4 Layers (bottom is near hardware, top is near user)
  + **Cocoa Touch**
    - The UI layer of iOS
    - Examples: Multi-touch, Buttons, sliders, Map Kit, Localization, Core Motion, Camera, Controls, Image Picker, etc.
  + Media
    - Video, Audio, Images, Animation,
    - Won’t be covered in this course
  + **Core Services**
    - OO layer on top of the Core OS
    - Examples: Collections, Core Location, Address Book, Net Services, Networking, File Access, SQLite, URL Utilities, Preferences, Threading
  + Core OS
    - It is a variant of UNIX
    - Everything down here is basically happening in C
    - Nothing will be taught down here
* Platform: Xcode 9
  + Source code, editing, debugging,
* Language: Swift (could use Objective-C, but Swift is newer)
* Frameworks = collections of objects: Foundation, Core Data, UI Kit, Map Kit, Core Motion
* Design Strategy (have to design this way): MVC = Model View Controller
* **Demo –** Concentration – matching 2 pictures by flipping them over, if they are a match remove the cards from the board, if not then turn them back over and try again
* Xcode
  + Create New Project:
    - Select Create new project button on Xcode home screen
    - Template for new project: “iOS” tab 🡪 “Single View App”
    - Product Name: name of the project
    - Team: yourself
    - Organization Name: anything
    - Organization Identifier: unique
    - Language: Swift
    - “Next”
      * Home Directory: cavanbriody 🡪 Developer
  + Main Project Editor Window
    - Left Sidebar = Navigator:
      * blue icon is the project
    - Right Sidebar = Utilities Pane:
  + Run Application
    - Top Left: Click on “Name of Project” > Select your Device to run it on (Simulator: iPhone X)
    - then click play (top left)
    - watch progress of build in the top center bar
    - Click the “Stop” button to end execution
  + Project Files:
    - Don’t need: make Supporting Files folder
      * Assets.xcassets: where all images, app icon are stored
      * LaunchScreen.storyboard:
      * AppDelegate.swift
    - Leave info.plist alone
    - Main.storyboard = UI
      * Make this UI graphically, not through code
      * Interface Builder
      * Can change the type of device you are looking at
  + Working on the UI in Main.storyboard:
    - Open up the Utilities Pane 🡪 Object Library (circle)
      * Drag any of the objects onto the phone display
    - Top of Utilities Pane 🡪 Attributes tab
      * Select a button to change its attributes or background screen to change its
    - Helpful: Edit 🡪 Emoji & Symbols
  + Working in ViewController.swift file:
    - Import UIKit
      * Brings all of UIKit in for our use
      * UIKit = iOS Framework that has buttons, sliders, etc.
    - class ViewController: UIViewController {
      * declaration of a class
      * *class* <Name of class>: *UIViewController {}* 
        + UIViewController is the superclass – inherited from UIKit
    - All methods and instance variables go inside the curly braces
    - Make a function to determine what is done when something is pressed
      * Open the Main and ViewController at same time with the two circles in the top right of the screen
      * Control + Drag the picture into the class in ViewController
        + Determine Connection type – Action

Outlet creates an instance variable

Outlet Collection – creates an Array instance variable

* + - * + Name the method
        + Arguments – Sender
        + Type: UIButton
    - func touchCard(\_ sender: UIButton) {}
      * function declaration – can be outside of the class (rare)
      * func <name>(<external param name> < internal param name>: <argument type>) -> <return type> {}
      * every argument has a name that you include when you call the method
      * each argument has 2 names – external (what callers use), internal (used inside the function implementation)
      * return value: -> Int
        + done after the list of arguments in the header)
    - print(“”) 🡪 prints a string to the console
    - don’t understand a function: option and drag over the function name – brings up the documentation of that func
    - **READ THE OVERVIEWS OF COMMON CLASSES**
    - Button Functions:
      * Button.currentTitle
      * Button.setTitle
      * Button.backgroundColor
    - Convert something to a string while printing/print a variable :
      * “this is a string \(<name of string>)”
    - var flipCount: Int
      * class instance variable/property: *var* <variable name>: <variable type>
      * need to have an initial value for the variable
      * Ways to initialize:
        + Initializer
        + = 0
    - Swift is strongly typed but has strong type inference
      * var flipCount = 0
      * type of flipCount is inferred as an Int
    - AVOID COPY AND PASTING
    - Property Observer:
      * Var flipCount = 0 {

didSet {

<code executed everytime the variable is set to something>

}

}

* + - Constant variables use “let” in the place of “var”
    - “nil” means an optional is not set
    - ! at end of optional means “assume that the optional is set and return its value
      * Or could use an if statement to check conditionally if the optionals are set
    - Rename a variable: Cmd + click 🡪 “rename”
    - ? is an optional variable – 2 potential values: set, not set
    - Nil = an optional variable is not set

**Lecture 2: *MVC***

* Model View Controller – way to divide the objects in the program
  + Model – **what** your application is but not how its displayed
    - Never speak to the View – Model is UI independent
    - Radio station – in order to communicate with the controller and then controller and tune into the station and fetch the updated data
  + View – the controller’s generic minions
    - Communication with Controller is blind and structured
      * Target action
      * Scrolling view – delegate – responds to “should”, “will”, “did” (delegate methods)
        + Delegate is set via a protocol
    - Do not own the data they display
      * Data source – use methods like “data at” and “count”
        + Controller is almost always the data source
  + Controller – **how** your Model is presented to the user – UI logic
    - Always able to talk to Model
    - Always able to talk to View – outlet
    - Primary purpose: interpret/format model information for the view – and vice versa
  + MVCs typically represent one screen in an app
    - Combine multiple MVCs in an app – one MVC is another MVCs “View”
* Struct and a Class are very similar in Swift
  + Differences:
    - No inheritance in structs – makes them simpler
    - Structs are value types and classes are reference types
      * Value type – gets copied
      * Reference type – passing pointers around to a single object
* Connect a Model (Concentration file) to the Controller (ViewController file)
  + In ViewController:
    - *var game = Concentration()*
      * Concentration initializer means there needs to be a init() in the Concentration file
* For loop
  + “for \_\_\_ in <sequence>”
    - A sequence is an array, string, range (“0..<10” or “0…10” including 10)

**Lecture 3: *Swift***

* Auto Layout