iOS App Development Bootcamp

Course Notes

# Section 1 – Getting Started with iOS 11 and Swift 4

## A Walkthrough of the Xcode Development Environment

* Core Data is used to save user data locally
* Click on a line number to insert a breakpoint

## The Anatomy of an iOS App

* Three main components:
  + View – what appears on screen
  + View controller – code that controls what happens
  + Model – controls the data
* Ex. Contacts app
  + View controller would ask model for data (phone number, name, etc.)
  + Model grabs data and passes back to view controller
  + View controller decides how information should be displayed
  + View displays information
* MVC
  + Can create new versions of app simply by swapping out data only
  + Makes it easier to debug

# Section 2 – iOS Interface Builder – The $999 App

## Let's Design the User Interface Using Storyboards

* Label resize to fit text shortcut -> Command =

## [How to Position User Interface (UI) Elements](https://www.udemy.com/ios-11-app-development-bootcamp/learn/v4/t/lecture/7556024?start=0)

* Top-left corner is origin (0, 0)
  + X value increases to the right, Y value decreases downward
* <https://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions> has screen resolutions for different iPhone models
* Object coordinates are marked by top-left corner of object

What's the Difference Between Points and Pixels & How to Design iOS Image Assets

* Pixel is a contraction of Picture Element
  + Each pixel is a single block of color
* Points
  + 1 point = 1/72 inch
  + Common resolution 72ppi (pixels per inch)
* App Icon naming convention: Icon-20@2x.png
  + Icon-20 = 20 points wide
  + @2x = screen resolution
    - @1x -> 1 point : 1 pixel (iPhone 4)
    - @2x -> 1 point : 4 pixels (2x2) (iPhone 5)
    - @3x -> 1 point : 9 pixels (3x3) (iPhone 6 Plus)
  + 1x images aren’t really used anymore

## Run and Test Your App!

* Always let Xcode open the app – don’t click the app yourself on the simulator

# Section 3 – Sideloading in iOS

What is Sideloading and How Can You Get Your App onto Your iPhone?

* Sideloading is building an app in Xcode and install it on your phone
  + Can do this for free without an Apple Developer Program account

## How to Sideload Your App using Xcode

* Xcode version must match iOS version
* Must set a team and allow Xcode to automatically manage signing
* Connect iPhone to Mac
* Choose your iPhone as build destination
* Trust profile on iPhone if necessary

# Section 4 – Coding Challenge #1 – Using Interface Builder in Xcode

## What is GitHub and How to Download the Stub Projects

* Github – a cloud-based service where programmers can upload code and share it with others
  + Based on Git technology
* Version Control is fundamental
  + Master copy is stored on server
  + Each person works on separate version
    - Can upload a version and merge it with master copy
  + If 2 people are working on a file and both try to merge with the master copy, the last person to merge will receive an error stating that there are conflicts between the two versions
    - That person can manually review the differences and do one of the following:
      * Override the master copy
      * Change his own copy
      * Edit the final version so that it incorporates parts of both files
* Can be used for multiple types of files (docx, pdf, etc.)
* Can keep an infinite history of previous versions

# Section 5 - Build an iOS Dice App to Learn Swift Programming Basics

## Linking Design with Code

* Storyboard is actually an XML file
* To link a storyboard object to your code, control + drag from the object to the code
* @IBOutlet = Interface Builder Outlet variable
  + Outlet changes the appearance of a UI element
  + Can hover over the line number circle to highlight the object in the storyboard
* @IBAction = Interface Builder Action function
  + Action lets the code respond when the user interacts with it
  + Can define which user interaction to look out for
    - Most cases it will be ‘Touch Up Inside’
      * User touches and lets go inside the object boundaries
* Buttons can use actions and outlets, but image views can only use outlets
  + Available options depend on the type of object

## [Common Errors] IBOutlets/IBActions Debugging

* Renaming/deleting an outlet/action after it has been created will unlink the storyboard object and the IBOutlet/IBAction
  + Renaming will make the line number circle show an empty circle rather than a filled one and will throw an error upon building the app
  + To correctly rename/delete these variables or functions, first right-click on the storyboard object and break the connection
    - If you renamed the variable or function, control-drag the object to the new variable name
  + Debugger error: “Terminating app due to uncaught exception ‘NSUnknownKeyException… this class is not key value coding-compliant for the key \_\_\_\_\_\_”

## How to Clean the Console in Xcode

* When running your app in a simulator, the console gets very cluttered and messy
* Xcode menu -> Product -> Scheme -> Edit Scheme
  + Run -> Arguments -> Add Environment Variable
    - OS\_ACTIVITY\_MODE -> value disable

Learn About Arrays in Swift and Programmatically Update the Display

* Change an object’s image: object.image = UIImage(named: String)

## [How to Become a Better Programmer] The DRY Principle

* DRY = Don’t Repeat Yourself
  + Creating “helper” methods can save you a lot of time and coding by combining repetitive or frequently executed lines of code into a separate function

## How to Add Motion Detection in to Your App

* override func motionEnded(\_ motion: UIEventSubtype, with event: UIEvent?) {}

# Section 8 - Apple iOS APIs - Make a Music App

## How to Use Tags in the Storyboards

* Can hook up multiple objects to a single IBAction
* Tag property
  + Allows you to differentiate between multiple objects if unsure of which object you’re dealing with

## How to Use the Apple Documentation to Tap into Any iPhone Capability You Want

* Utilize stackoverflow to find how to implement various functionality
  + Also use forums.developer.apple.com
* Hold down the option key and click on items in code to show documentation
* Hold down the command key and click on items in code to access library code
* Apple API reference can be used to find reference documents for lots of things
* Good practice to use these references to find out more about code blocks that you find online

## Incorporate AVFoundation to Play Sound

* Import AVFoundation
  + Allows us to utilize audio components of the iPhone

# Section 9 - The Model-View-Controller (MVC) Design Pattern - Make a Quiz App

## Set up the Project

* Can edit the plist file to change the style of the status bar
* Good practice to make the default text of a label to be the highest possible digit count

## Let's Create a Data Model

* A method is a function that is associated with a class
* A function is not associated with a class
* Initializer/constructor: init(…)

## Object-Oriented Programming - What are Objects and Classes?

* Procedural programming languages
  + A program is a single list of instructions
  + Very linear
* Object oriented programming
  + Multiple objects that can send messages to each other
  + Examples:
    - Objective-C
    - Java
    - Swift
* Class
  + Blueprint
  + Instructions for creating objects
* Objects
  + Thing that gets created from a class
  + Have properties
    - Constants or variables declared inside a class
  + Have actions/functions
    - Functions inside of a class are called methods
  + Have events
    - How the object responds to things happening to it
    - Init() is an event

## [ELI5] The Model View Controller (MVC) Design Pattern - Explain Like I'm 5

* Design pattern: A proven solution to a common problem
* Model
  + Structure that manage the data
  + Read/writes to/from the database
  + Not the data itself
* View
  + What you see on screen
* Controller
  + Handles communication between model and view
* MVC:
  + Provides a structure to your code
  + Recognizable by third parties
  + Code reuse
  + Allows for multitasking

## How to Implement a UIAlertController and Show a Popup to the User

* UIAlert preferredStyle:
  + Modal – many options to choose from
  + Action Sheet – Pops up from bottom of screen

# Section 11 – iOS Auto Layout and Setting Constraints

## Positioning UI Elements Programmatically

* Examples:
  + X = self.view.frame.width / 2
  + Y = self.view.frame.height / 2
  + Width = 100
  + Height = 100
  + Centered on screen:
    - X = self.view.frame.width /2 – square.width / 2
    - Y = self.view.frame.height /2 – square.height / 2

## The Theory Behind Auto Layout

* Can set constraints through:
  + Pinning
    - Each edge of the object should be x number of points away from the respective side of the screen
    - Object height and width expands and contracts to match pinning requirements on different screen sizes
    - Objects will be distorted when changing device orientation
  + Alignment
    - Keeps the width and height constant
    - Allows you to keep object in a particular place on the screen
    - Objects will retain shape when changing device orientation
* Depending on method, you must provide (x and y) or (width and height)
  + Pinning requires x and y
    - Width = screen width – 2x
    - Height = screen height – 2y
  + Alignment requires width and height
    - X = (screen width / 2) – (shape width / 2)
    - Y = (screen height / 2) – (shape height / 2)

## Auto Layout in Practice – Setting Constraints

* Order of operations matters!
* Pinning
  + Box-and-whisker plot
    - Uncheck Constrain to Margins FIRST
      * This means that we want to set the constraints to the edges of the screen, not on the margins which are slightly inside the edges
    - On dropdown window for each number, you can choose which object to be constrained to. Typically, you will want to choose View
* Alignment
  + Box-and-whisker plot
    - Check width and height boxes to keep them static
  + Bar graph plot
    - Check Horizontally in Container and Vertically in Container and set them to 0 to keep the object in the middle of the screen
* To constrain multiple objects together, put them in a UIView container
  + Create a new view
  + In the Scene navigator, drag the child objects to the child position of the view
  + Need to constrain child objects to the view container as well as constraining the container to the super view
    - Often times you will want to center the child object in the container view, so follow the steps above to do that
  + Give each container view a different color to stand out from the other containers as well as to see how the containers look on each type of phone
    - When you’re finished with constraints, set the container views’ background color to default (transparent)
* **IMPORTANT**: You MUST make sure that you have (height and width) or (x and y) values specified. Whenever an error pops up, make sure that you know what each value is/how it should be filled.

## Auto Layout Using Stack Views

* Useful for when you have a grid system of objects on screen
* To use, select multiple objects and click the Embed in Stack button to the left of the constraints buttons
* Can use the Alignment and Distribution settings in the attributes inspector to adjust fill
  + Used Distribution: Fill Equally in the example
* Can then use constraints to stretch across the screen
* Adjust the Spacing property as needed
* Can put stacks in stacks

# Section 12 – Programming in Swift 4 for Intermediates

## Creating a New Class from Scratch

* Class names must be capitalized

## Creating Enums in Swift

* Allows you to create a custom data type with different options
* Enum names must be capitalized
* Similarly to a switch block, enums use ‘case’ notation with no actions
  + Ex.
  + enum CarType {
  + case Sedan
  + case Coupe
  + case Hatchback
  + }
* Create a new type by using dot notation
  + Ex.
  + var typeOfCar : CarType = .Coupe

## Class Initialization

* init() is used to create an initializer

## Designated and Convenience Initializers

* Designated initializers have parameters
* Convenience initializers are designated by the keyword ‘convenience’ and also take parameters
  + The first thing run is init()
  + Allows you to create custom objects for very specific use cases
  + Can customize specific properties while leaving the rest default

## Class Inheritance – The Theory

* Syntax: class SelfDrivingCar : Car {}
  + SelfDrivingCar is a subclass of the Car superclass
* Subclasses inherit variables, initializers, and methods of superclasses
* Can override inherited superclass methods using the override keyword
  + Syntax: override func doSomething() {}
  + Typically want to call the super.doSomething() first in the overridden method

## What are Swift Optionals?

* Optionals are variables that can possibly contain the value nil
* Syntax: var myVar : String?
* Syntax: var myVar : String!
* nil doesn’t equal the empty string
* Force Unwrapping: When you call a variable with an exclamation point, you are telling xcode that you know that the variable will NOT be nil
  + Xcode converts the optional to a non-optional variable
* When you call a variable with a question mark, Xcode will be prepared to throw an error if it is nil
* Safety check - check if variable != nil
  + Optional binding:
    - if let userSetDestination = destination {
    - print("Driving towards " + userSetDestination)
    - }
    - Benefit: Don’t need to force unwrap anything (force unwrapping uses !)
  + Optional chaining:
    - delegate?.userEnteredANewCity()
      * If variable delegate is not nil, it executes the method
      * If it is nil, nothing happens

# Cocoapods, GPS, APIs, REST and Getting Web Content - Weather App

## What are Cocoapods and why do we need them?

* Cocoapods is a dependency manger for Xcode
  + Is a library for portions of code written by others that you can use in your projects
  + Cocoapods.org – can search for any pod to use
  + Cocoapods is a ruby gem
* Installing cocoapods:
  + sudo gem install cocoapods
  + pod setup --verbose

## Installing New Pods in Your Xcode Project

* Close Xcode
* In terminal, cd to Xcode project folder
* pod init
* open –a Xcode Podfile
* Delete the comment symbol on line 2
* Add the following lines under # Pods for Clima:
  + pod 'SwiftyJSON'
  + pod 'Alamofire'
  + pod 'SVProgressHUD'
* Save and close Podfile, quit Xcode
* pod --version
  + make sure you have version 1.1.0 or above
* pod install
* Note: Can run pod update to update all pods in project
* IMPORTANT: You must open the Xcode project from the .xcworkspace file from now on rather than the .xcodeproj, as it contains the pod code

## Setting up the Location Manager to Get GPS Data from the iPhone

* [Extras]
  + To use weather data from openweathermap, you’ll need to sign up for a free account and request your own api key
  + Pragma mark: allows you to catalog your sections of code for easy navigation from the breadcrumb dropdown menu
    - Syntax: // MARK:
    - Syntax: // MARK: -
      * Puts following text in bold in menu
* [/Extras]
* First, import CoreLocation library
* Need to make class a delegate of CLLocationmanager
  + Syntax: class myClass: CLLocationManagerDelegate {}
  + This means that our class conforms to the rules of the CLLocationManagerDelegate protocol
* Create CLLocationManager object
* Set above object.delegate = self in viewDidLoad()
* Note: Your view controller must become a delegate to use this location api. This makes it so that the location manager knows who to report to when it acquires its data. Delegate/protocols are an intermediary concept in swift.
* Set location manager desired accuracy
  + Syntax: locationManager.desiredAccuracy = kCLLocationAccuracy…
* Ask user for permission to use location data
  + Syntax: locationManager.requestWhenInUseAuthorization()
  + .requestAlwaysAuthorization uses location even when app is in background

## Asking the User for Location Permissions

* In order to request permission, you must edit the message in the .plist file
* Add the following to the plist:
  + ‘Privacy – Location Usage Description’
  + 'Privacy – Location When In Use Usage Description’
  + The values should be the same message that is displayed to the user
  + Can also edit as XML source code
* Add Fix for App Transport Security Override code from project github readme right below NSLocationUsageDescription string
* Pop up should now work!

## Tapping into the GPS

* Under where you set up the location manager, call the location manager’s startUpdatingLocation() method
  + This is an **asynchronous** method, which means it runs in the background (not on the main thread) independent of other running code
* Because the view controller is the delegate, the location manager will return the user’s location, but it has to do it through the locationManager(manager, location) method
  + Passes locations through a CLLocation array
    - The most recent item should be the most accurate
  + Need to check if the location’s horizontalAccuracy > 0
    - This is a circle that contains the user’s possible location and the radius is the degree of accuracy
  + If you have a valid result, call the location manager’s stopUpdatingLocation() method do that you aren’t draining the user’s battery
    - It can take a bit of time for this method to complete; you can set the locationManager.delegate to nil to stop receiving responses while the stopUpdatingLocation() method is running
  + Create a dictionary according to the openweathermap API to hold the latitude, longitude, and app ID
* Also need to implement locationManager(manager, didFailWithError) to catch errors retrieving data

## Delegation

* If you want to pass data between classes, you can create an object of one of the classes to access its data
* You can’t do this with Apple-created classes
  + Their classes were created long ago and they wouldn’t have written code to create an object for one of your classes
* This is where delegation comes in
  + Our classes can be intermediaries
  + Apple-created classes can send their information to a delegate

## What are APIs?

* Application Program Interface
* A contract that specifies the rules for accessing the data from the external app or server
* You can make a program access a website normally to find the information it needs, but if the website structure changes, your program will break
  + APIs are a way to get around this
* APIs are more robust and easier to use
* Lots of websites have APIs available

## Networking with Alamofire

* When you need to access web data, you must make an http request to get a response
* Will use Alamofire to make requests
  + Can look up documentation on alamofire github page
* Standard Alamofire http request:
  + Alamofire.request(url, method: .get, parameters: parameters).responseJSON {
  + response in
  + if response.result.isSuccess {
  + }
  + else {
  + }
  + }
  + Breakdown:
    - Takes in the url we give, uses the get request method, and uses the parameters from our dictionary to create a legitimate request
    - Checks if the request successfully grabbed data
    - Allows for a response if no data was grabbed
    - Happens asynchronously (in the background)

## Networking and REST Explained

* Networking: Communication between computers on a network
  + Ex. Browser makes a request to server; server returns html and css files; browser interprets and displays files
  + Browser passes along http request with a URL
* HTTPS request methods
  + GET: Fetching data
  + POST: Pass data to server/add data to server
  + DELETE: Request to delete data in server
* Clima app makes GET request to openweathermap API; server gives us response with data
* Alamofire request method takes a URL, request method, and parameters
* Response data is held in the response variable

## What is JSON and How to Parse it

* JSON: Javascript Object Notation
* JSON is a common way of formatting large amounts of data on the internet
* JSON integration comes from SwiftJSON pod
* For Clima, we need to create a JSON object to contain the weather data since that is what the data type of the openweathermap response is
  + Need to force unwrap it first, but we can confidently do this because we are already checking to make sure that there is a successful value returned in the response
  + Also need to cast it to a JSON type
* Can use joneditoronline.org to display JSON data in object format
* Most API responses are formatted in JSON
* Alamofire request method contains a closure
  + Closures are designated by the keyword ‘in’
  + Closures are basically functions within functions
  + To call the updateWeatherData to parse the JSON string, we have to call it using self because we’re in a closure and the compiler can’t quite figure out where we’re trying to call it from
* SwiftyJSON makes parsing through JSON objects much easier and more efficiently

## Create a Weather Data Model

* We could go through property by property and update the individual labels separately, but this is considered bad practice
* Instead, we will create a data model that holds all of the data we need and will use that model to update everything at once
* When updating the data model, be sure to check if the data is valid
  + If not, let the user know in a friendly manner

Creating and Using Segues in Xcode

* Segues allow us to link view controllers to other view controllers or elements in other view controllers
* Clicking on a segue highlights its origin, be it a button or view controller
* To set up a segue:
  + CTRL + drag from button/source to new view controller
* Different types of segues:
  + Show – view controller slides onto current view controller
  + Experiment with others
* Need to create a new swift file (Cocoa Touch Class) for the new view controller
  + Change the view controller class in the identity inspector to your new class
* Can also call segues in code
  + Create the segue by dragging from the yellow circle at the top of the first view controller to the inside of the second view controller
  + Need to give your segue an identifier in the identity inspector
  + Call the segue in your code with the function performSegue(withIdentifier: String, sender: Any?)
    - Can use self for sender
* Can pass data from first view controller to second
  + override func prepare(for segue:…) {
    - if segue.identifier == “identifier” {
      * Let destinationViewController = segue.destination as! SecondViewController // <-- this is the class of the second view controller
      * destinationViewController.textPassedOver = textField.text! // textPassedOver is a String created in the second view controller
    - }
  + }
  + You can then set the text property of the label in the second view controller to equal textPassedOver in the viewDidLoad() method, since this won’t be called until the segue is called
  + Note: You will need to create a specific case for each identifier in the overridden prepare method
* To dismiss the screen, call self.dismiss(animated: true, completion: nil)

## Using Delegates and Protocols in iOS App Development

* Protocols are created outside of classes
* There seem to be two classes used with a protocol:
  + A class that is associated with a protocol
    - Ex. CLLocationManager is associated with CLLocationManagerDelegate
  + A class that is to be a delegate of a protocol
    - Ex. WeatherViewController is the delegate of CLLocationManagerDelegate
* If you want a class to be a delegate of a protocol, that class must implement any methods of the protocol and have the protocol after the class name (class MyClass: Protocol)
  + This class must also create an object that is of type “class associated with the protocol”, and then set that object’s .delegate property equal to self
    - When using segues, this variable must be declared as follows:
      * Let variable = segue.destination as! “name of associated class”
* In the class associated with the protocol, there must be a variable created (typically named delegate) that is of type “name of your protocol”?
  + Needs to be an optional because the value of the delegate property can be nil
  + This variable can call the protocol methods (which should be implemented in the class that is a delegate of the protocol), because by the time these methods get called, the class that is a delegate of the protocol should have already set the delegate property to self
* Note: For Clima, we can’t just create a variable to contain this data because the WeatherViewController class itself needs to have the information from ChangeCityViewController’s text field. If we just created an object of WeatherViewController to pass the data to, that object not be the same view controller that you segue to. To get the actual view controller that you are segueing to, you must access it with segue.destination. Delegation is a higher level of passing data directly between classes that doesn’t rely on objects.

## How to Pass Data Between View Controllers

* If you pass data back and forth by only using prepare(for segue: …), you’ll only be creating new objects with each segue
* Need to use protocols, delegates, and the dismiss function to truly pass data back
* Protocols are at the same hierarchal level as classes and can be created in their own swift file, but that isn’t required
* Rough workflow of protocols and delegates:
  + Protocol is created with some methods
  + Delegate class adds protocol name to class declaration
  + Delegate class implements protocol methods
  + Associated class has a property named delegate that is of type “name of your protocol”? (this variable is an optional)
  + Associated class calls protocol methods using the delegate? variable
  + Delegate class creates a variable of type “associated class” using segue.destination
  + Delegate class sets above variable’s delegate property to self
  + Second view controller is dismissed

# Section 14 – The Command Line and Terminal

## Command Line Shortcuts

* ls – list
* CTRL + A – go to beginning of line
* CTRL + E – go to end of line

## Directory and File Manipulation Using the Command Line + Bonus

* mkdir – create new directory
* touch <filename.ext> - create new file
* open <filename.ext> - open file
* open –a <application> <filename.ext> - open file in application
* rm <filename.ext> - remove file
* rm \* - remove all files in current directory
* rm -r <directory name> - remove directory recursively
* Bonus: emacs –batch –l dunnet

# Section 16 – Cloud Database, iOS Animations and Advanced Swift – Make a WhatsApp Clone

## What is Firebase?

* A service from Google that provides a cloud-based storage service that allows us to store and retrieve data
  + Has a bunch of features
  + Free to use
  + Can have many users before needing to pay
* We will be using real-time databases
* We will also be using authentication with email/password

## Set Up Firebase for Development

* Log into Firebase and click on ‘Go To Console’
* Click iOS project
* Register app with bundle identifier, then download .plist file and drag it into your xcode project window under the existing .plist
* Install the firebase cocoapod, as well as any other cocoapods you need
* Copy Podfile configuration from github page into podfile
* Close xcode and run pod update

## Saving Data to the Database

* Used the following lines to configure the database and send some test data:
  + FirebaseApp.configure()
  + let myDatabase = Database.database().reference()
  + myDatabase.setValue("We've got data!")

## Let’s Register Some Users To Our App

* On the Firebase dashboard, go to develop -> authentication -> sign-in method
  + Can choose many different methods of signing in; try to implement google and facebook later
* In the RegisterViewController, you’ll need to implement the Auth.auth().createUser(withEmail: emailTextfield.text!, password: passwordTextfield.text!) function
  + The closure code should look like this:
    - {
    - (user, error) in
    - if error != nil {
    - print(error!)
    - }
    - else {
    - print("Registration successful!")
    - }
    - }
  + Note: The code above is a closure. Since the closure is the last parameter of the function, it can be written in brackets outside of the input list
  + Firebase detects for correct email and password formatting
* Can also add users in Firebase Dashboard
* Can create a password text field by checking the ‘Secure Text Entry’ field in the attributes inspector
* Need to perform segue in the else statement above, but must use self.preformSegue(…) because it’s in a closure
* Closure:
  + Functions without name
  + Equivalent to a function within a function
  + Usually designated by the ‘in’ keyword

## Swift Closures

* Self-contained package of functionality
* You can pass a function as an input and return a function as a return value
* Can pass it formatted like so:
  + name: (input1, input2) -> output
  + Can call the function by its name inside of the original function, just like a variable
  + When you call the original function, you can pass it the name of an existing function that meets the input and output requirement
* To convert a function to a closure, remove ‘func’ and the name of the function, move the opening brace to the beginning of the code block, and insert the ‘in’ keyword right after the return type
  + Can then use this whole closure as an input parameter
  + Can also delete closure input data types, as well as the entire ‘-> type’ output statement
  + Can ALSO get rid of return keyword
  + Can ALSO replace input parameter names with anonymous parameter names
    - $0 for first parameter
    - $1 for second parameter
    - So on and so forth
  + Can ALSO get rid of ‘in’ keyword when using anonymous parameters
  + Can have a closure boil down to operation: {$0 \* $1}
* If the closure is the last parameter, you can omit the name for the closure and just use the braces, like this:
  + Let result = calculator(n1: 2, n2: 3) {$0 \* $1}
* Array.map takes a function as a parameter and applies that function to every item in the collection
  + Can also cast items to different data types
* General closure syntax:
  + { (parameters) -> return type in
    - statements
  + }
* Can put the contents of the closure on a new line to basically make it look like a function

## The Event-Driven Nature of Apps

* Revisiting the anatomy of an app
* We have a button and a view controller
  + Button click tells the view controller to update/do something
* We write code in anticipation of events
  + Events can be triggered by OS, third party APIs, or users
* The App Lifecycle
  + App launch
  + Becomes visible
  + Recedes into background
  + Resources reclaimed (app killed)
* Completion handler/callback
  + Functionality that gets triggered upon the completion of an event

## What are Completion Handlers and How Do They Work?

* Completion handler: a function takes in a function as a parameter and runs after the main function completes
  + If the parameter function returns nothing, it must return Void
  + Can call this parameter method by variable name inside the main function
* When calling a function that has a completion handler, just use the function name without any parentheses or parameters for the completion handler
  + Can also pass a closure rather than creating a new method following the steps above to create it

## How Do Navigation Controllers Work and How Do You Create Them?

* All of the Flash Chat View Controllers are nested in a navigation controller
* To set up a navigation controller, select the initial view controller, go to Editor -> Embed In -> Navigation Controller
* Different types of segues
  + Show: Typically used for view controllers that are nested inside a navigation controller
  + Present Modally: The view controller that gets presented is not part of the navigation stack
    - Doesn’t get the navigation bar
    - Pops up from bottom
* Can use navigationController?.popToRootViewController(animated: Bool) to go back to the root view controller
  + The root view controller is the initial view controller, as well as the bottom layer in the navigation stack
  + For Flash Chat, we use this method for the log out button to go back multiple layers to the welcome screen

## What are TableViews and How to Set One Up?

* TableView: a view with cells that are scrollable
* Can drag a tableview onto the storyboard to create a new table view controller, or drag it into an existing view controller to embed it
* Need to conform class to UITableViewDelegate and UITableViewDataSource
* Using the IBOutlet variable for the UITableView, set the delegate and dataSource properties equal to self
* Need to implement protocol methods numberOfRowsInSection and cellForRowAt
* TableViews generate enough cells to fill the screen
  + When you scroll and cells go off screen, the cells get recycled and are used as the new cells that appear on screen
* In cellForRowAt, we need to create a new cell using tableView.dequeueReusableCell
  + Takes an identifier; we can use the custom cell identifier
  + Takes an index path
    - This is basically a location identifier that says where a cell should be
  + Must append as! <custom cell class> if using a custom cell
* To create a new custom cell, create a new cocoa touch class file, make it a subclass of UITableViewCell, and check the box for ‘Also create a Xib file’
  + The xib file is where you can design how you want the cell to look
* You’ll want to create outlets from the xib file to the new custom cell class to access the components of the cell
  + To use the custom xib, you need to register it with the table view controller
    - messageTableView.register(UINib (nibName: "MessageCell", bundle: nil), forCellReuseIdentifier: "customMessageCell")
* Can manually adjust cell components such as text, etc.
* Then need to return the cell
* For numberOfRowsInSection, you need to return an int for the number of cells you want
* Can also implement the numberOfSections and return an int for the number of sections in the view
* Standard cell height is 44pt, so you can change it through a new method and some contraints
  + func configureTableView() {
  + messageTableView.rowHeight = UITableViewAutomaticDimension
  + messageTableView.estimatedRowHeight = 120.0
  + }
  + When your estimated row height is not correct, UITableViewAutomaticDimension will use the constraints set up in your custom cell xib

## Making a Custom Message Class

* You need a model to hold the data for the messages
  + This model will contain a variable for the message sender as well as a variable for the message body

## Learn About UI Animations in iOS and Make Our Own

* Tween animation
  + Specifying the first and last frames in the animation and deciding how much time it should take to transition between them
  + Computer will calculate and create necessary amount of in-between images
* Need to create an animation to move the chat text box higher in the screen as to not be obscured by the keyboard
  + Will use an IBOutlet connected to the height constraint of the text box
* Need to detect when the user taps into the text box
  + Need to make ChatViewController a UITextFieldDelegate
  + Set the text field delegate to self
  + Implement the optional methods textFieldDidBeginEditing and textFieldDidEndEditing which tell the ChatViewController when the focus is on and off of the text box
* In the above methods, call the UIView.animate(withDuration)
  + In the above closure:
    - Set the heightConstraint.constant property to the height required
    - Call with view.layoutIfNeeded() to update the autolayout constraints
    - Must use the self keyword before both of these because we’re in a closure
* textFieldDidBeginEditing gets triggered automatically, but we have to call textFieldDidEndEditing manually
  + Must create a custom tap gesture recognizer in the tableview to detect when the user taps outside the text box

## Let’s Send Some Messages

* [All code takes place in the sendPressed @IBAction for the send button
* Clicking the send button should end the editing in the text box and collapse the keyboard
  + Call messageTextField.endEditing(true)
* The send button and text box should be disabled during the networking process to duplicate messages aren’t sent
  + Set the messageTextfield and sendButton isEnabled property to false after the networking is invoked
* Need to create a variable to be the Firebase database for the messages
  + let messagesDB = Database.database().reference().child("Messages")
* Need to create a dictionary that will store the sender and message information in a key-value pair
  + let messageDictionary = ["Sender" : Auth.auth().currentUser?.email, "MessageBody" : messageTextfield.text]
* Can add a messageDictionary to the Messages database using childByAutoID. This adds an element with a randomly generated unique ID so that each message will have a different identifier. We can reenable the send button and text field, as well as clear out the message text field
  + messagesDB.childByAutoId().setValue(messageDictionary) {
  + (error, reference) in
  + if error != nil {
  + print(error)
  + }
  + else {
  + print("Message saved successfully")
  + self.messageTextfield.isEnabled = true
  + self.sendButton.isEnabled = true
  + self.messageTextField.text = “”
  + }
  + }

## Observing for Database Changes

* Need to create an instance variable to hold an array of messages
  + var messageArray : [Message] = [Message]()
* Need to create a new function to listen for new entries to the database
  + func retrieveMessages()
  + Create a variable that holds the same database reference to when you sent messages earlier:
    - let messageDB = Database.database().reference().child("Messages")
  + Use Firebase API to observe changes
    - messageDB.observe(.childAdded) { (snapshot) in
    - let snapshotValue = snapshot.value as! Dictionary<String, String>
    - let text = snapshotValue["MessageBody"]!
    - let sender = snapshotValue["Sender"]!
    - let message = Message()
    - message.messageBody = text
    - message.sender = sender
    - self.messageArray.append(message)
    - self.configureTableView()
    - self.messageTableView.reloadData()
    - }
      * .observe looks for added children to the database and has a closure that allows you to read a snapshot of the database
      * We know that the snapshot (which is type Any?) is a Dictionary because we are the ones who provided the data
        + Type Any? Is legacy from objective-c and is used to help support swift when interacting with objective-c code
      * Need to append the new message to messageArray that we defined earlier
      * Call configureTableView() again to readjust the cell heights
      * Call messageTableView.reloadData to update the UI with the new data
* Call retrieveMessages() at the end of viewDidLoad()
* Need to edit tableView(cellForRowAt) to replace the dummy data with the real data
  + cell.messageBody.text = messageArray[indexPath.row].messageBody
  + cell.senderUsername.text = messageArray[indexPath.row].sender
  + cell.avatarImageView.image = UIImage(named: "egg")
* Need to change tableView(numberOfRowsInSection) to return the count of the messageArray

## Bug Fix – Where’s the Historical Message Data?

* When the app gets relaunched, the entire database gets wiped
* In the AppDelegate.swift, we are setting the value of the entire database to “We’ve got data!”
* Comment out this code and the database will persist

## Update the Security Setting of Your Database

* Currently, anybody can read/write to the database
* Click the “Find Out More” button to go to the Database Rules documentation
* Change the read and write rules to “auth != null”

## Let’s Improve the UX – Add a Progress Spinner

* Will be using SVProgressHUD
* At login and register events, use SVProgressHUD.show() and .dismiss() to show and hide the loading wheel

## Improving the UI – Differentiating Between Different Users

* To get rid of the message separators, call messageTableView.separatorStyle after viewDidLoad() in ChatViewController
* We want a different style for different users
* Will use ChameleonFramework
  + First import it
  + Can use UIColor.\_\_\_\_\_\_ to find colors

# Section 18 – Git, GitHub and Version Control

## Version Control Using Git and the Command Line

* To initialize git, run git init in the desired directory
* You first need to add files to a ‘staging’ area
  + Use git add <filename>
  + Can use git status to see what files are in the working directory that has uncommitted changes as well as what files are in the staging area (added)
  + Can use git add . to add everything that has been changed to the staging area
* Can the commit using git commit -m “commit message”
  + Should always use present tense in commit messages
* Can see commits using git log
* Can use git to roll back files
  + Use git diff <filename> to view the diff
  + Use git checkout <filename> to roll back file