Become an iOS Developer Nanodegree Program Syllabus

Build an App for the iPhone and iPad

BEFORE YOU START

The journey to becoming an iOS developer begins in your imagination—that moment when you first dream up a great idea for an app. This Nanodegree program will prepare you to publish your first iOS app, whether you're already programming or just beginning. As you master the Swift programming language and create a portfolio of apps to showcase your skills, you'll benefit from detailed code reviews, valuable career advice, and coaching from professional iOS developers.

Prerequisites:

- You are self-driven and motivated to learn. Participation in this program requires consistently meeting deadlines and devoting at least 10 hours per week to your work.
- Collaboration with peers and interactive feedback are critical to the success of the program. You must be a committed and contributing participant of the community.
- Access to a Mac computer running macOS 10.12.6 or later

Educational Objectives:

Students will master writing iOS apps in Swift as they build six portfolio-worthy apps to demonstrate their expertise as an iOS Developer

Length of Program*: 6 months
Frequency of Classes: The program is flexible, self-paced with suggested project deadlines
Textbooks required: None
Textbooks optional: (if applicable)
Instructional Tools Available: Video lectures, mentor-led student community, forums, project reviews

*The length of this program is an estimation of total hours the average student may take to complete all required coursework, including lecture and project time.

If you spend about 10 hours per week working through the program, you should finish within the time provided. Actual hours may vary.

Learn Swift Programming (Optional)

You will complete a series of coding exercises to test your understanding of Swift. There will be exercises for variables, strings, if (else-if and else) statements, and functions.

Supporting Lesson Content: Learn Swift Programming

Lesson Title	Learning Outcomes
Variables and Types	➔ Declare variables and constant values with basic Swift types like Bool, Int, Double, and Float ➔ Access and modify values from variables and constants ➔ Debug compiler issues related to the incorrect use of variables and constants ➔ Use escape characters and string interpolation to format variable and constant values within strings.
Operators and expressions	➔ Compute new values using existing variables and constants. ➔ Use comparison operators to determine equality between two values. ➔ Use boolean operators to build expressions that use truth values.
Control Flow	➔ Write boolean expressions that convey decision making logic ➔ Combine boolean expressions with logical operators ➔ Utilize boolean expressions alongside if, else-if, and else statements to control the flow of your code's execution. ➔ Use switch statements to run code based on multiple values of a single variable. ➔ Use for, while, and repeat while loops to control the flow of your code's execution.
Functions	➔ Encapsulating existing code into reusable functions. ➔ Properly define and call functions. ➔ Specify function parameters and return types. ➔ Differentiate between values that are in-scope and out-of-scope. ➔ Correctly use local and external parameters. ➔ Identify parameter types and return types.
Structures and Enum	➔ Group multiple values together into structs. ➔ Create instances of structs. ➔ Add functions (known as methods) to structs. ➔ Access properties and call methods of structs. ➔ Define computed properties that calculate their value based on

	other values. ➔ Define enums and assign raw values to different cases. ➔ Use enums in conjunction with switch statements.
Optionals	➔ Understand when a value can be nil and when to use an optional type. ➔ Declare variables and constants as explicit or implicitly unwrapped optionals. ➔ Unwrap optionals both safely and unsafely. ➔ Use optional chaining and the nil coalescing operator to safely access optional values.
Strings	➔ Define and manipulate Strings using their built-in properties and methods ➔ Perform common String operations like concatenation and finding substrings ➔ Perform common String manipulation such as adding, removing, and replacing substrings.
Collections	➔ Store unordered data of the same type using arrays. ➔ Access and modify array contents. ➔ Store pairs of keys and values using dictionaries. ➔ Access and modify dictionary contents. ➔ Store unordered data of the same type using sets.
Object Oriented Programming	➔ Understand the difference between value and reference types, and how this applies to structs and classes. ➔ Make one class inherit the properties and methods of another class. ➔ Understand polymorphism - how one type can be substituted for another type, and how this relates to inheritance. ➔ Write classes that conform to the same protocol. ➔ Add additional functionality to classes using extensions.

Project 1: Pitch Perfect

You will create an iPhone app that records audio and plays it back using various audio filters and modes including adjusted rate and pitch, echo, and reverb.

Supporting Lesson Content: Intro to iOS App Development with Swift

Lesson Title	Learning Outcomes
Introduction and Xcode	➔ Navigate the major components of the Xcode development environment including the Navigator, Debug Area, and Utilities ➔ Create an Xcode project for a new iOS application ➔ Express the goals and architecture of the Model View Controller (MVC) design pattern

AutoLayout and Buttons	➔ Use Storyboards, Xcode's visual editing tool, to position, size, and configure user interface objects ➔ Link user interface objects in a Storyboard to their corresponding controller using IBOutlets ➔ Specify callback functions called IBActions that are invoked as a result of user interaction ➔ Create AutoLayout constraints to ensure UI elements are sized and positioned correctly regardless of device size and dimensions
ViewController and Multiple Views	➔ Configure application state at the appropriate customizations points in a view's lifecycle ➔ Create and navigate multiple-view applications using a UINavigationController ➔ Manipulate user interface objects by utilizing IBOutlets and IBActions
Delegation and Recording	➔ Write protocols to express functionality that can be adopted by Swift classes ➔ Use protocols to delegate the responsibilities of a particular task or set of tasks to another object ➔ Create and interface with an AVAudioRecorder to capture and save audio with an iOS device's microphone ➔ Use segues to transition between views in an application
Playback and Effects	➔ Create and configure StackViews which contain and automatically configure layout constraints for its subviews ➔ Playback audio using objects defined in the AVFoundation framework ➔ Apply audio playback effects using audio nodes exposed by a custom interface
Suggested Electives	➔ Version Control with Git ➔ GitHub & Collaboration

Project 2: MemeMe 1.0: The Meme Editor

You will create a first version of the MemeMe app that enables a user to take a picture, and add text at the top and bottom to form a meme. The user will be able to share the photo on Facebook and Twitter and also by SMS or email.

Supporting Lesson Content: UIKit Fundamentals

Lesson Title	Learning Outcomes
Outlets and Actions	➔ Understand how to connect outlets and actions using only code and graphically using storyboard

	➔ Use core UIKit classes like UIButton, UILabel and UISwitch. ➔ Practice debugging problems with IBOutlets and IBActions
View Presentations and Segues	➔ See how Apple distinguishes between modal presentation and navigation. ➔ Learn how to present views modally. ➔ Use powerful UIKit classes like UIImagePickerController, UIAlertController and UIActivityViewController.
The Delegate Pattern	➔ Learn how delegates make important connections between the model, view, and controller. ➔ Implement UIKit components that make use of the delegate pattern, UITextField and UITextFieldDelegate. ➔ Demonstrate your understanding by building a series of challenge apps.

Project 3: MemeMe 2.0: The Final Product

You will create an app that enables a user to take a picture, and add text at the top and bottom to form a meme. The user will be able to share the photo on Facebook and Twitter and also by SMS or email. Memes will appear in a tab view with two tabs: a table view and a collection view.

Supporting Lesson Content: UIKit Fundamentals

Lesson Title	Learning Outcomes
Table Views	➔ Learn the essential UITableViewDelegate and UITableViewDatasource methods. ➔ Explore the code for several apps with tables, and then implement your own UITableView. ➔ Practice manipulating table cells.
Navigation	➔ Learn how iOS uses navigation stacks to manage multiple views in an app. ➔ Create the navigation that enables a user to tap a row of a table and view the details of an item. ➔ Learn navigation classes like UINavigationControll and UIBarButtonItem.
Suggested Electives	➔ AutoLayout

Project 4: On the Map

You will create an app with a map that shows information posted by other students. The map will contain pins that show the location where other students have reported studying. By tapping on the pin users can see a URL for something the student finds interesting. The user will be able to add their own data by posting a string that can be reverse geocoded to a location, and a URL.

Supporting Lesson Content: Network Requests and GCD

Lesson Title	Learning Outcomes
Making a Network Request	➔ Express the flow of data from a client to a server when a client makes an HTTP request ➔ Create a network request in Swift and receive and consume a data response ➔ Switch execution from a background thread to a (main) foreground thread to avoid blocking an app's UI ➔ Abide by Apple's App Transport Security protocol to ensure user safety when access data over a network ➔ Download and display an image using a simple network request
Using Web Services and APIs	➔ Make requests to a web service (API) using documented endpoints and parameters ➔ Make a GET request to access data stored on a remote server ➔ Use a web service to download JSON data ➔ Convert raw byte data into JSON-like data that can be consumed by an app
Problem Set: JSON Parsing	➔ Extract values from JSON objects and arrays ➔ Access data from a locally defined JSON file
Chaining Asynchronous Requests	➔ Perform multiple network requests in sequence using callbacks and closures
Authenticating Requests	➔ Perform an authorization flow that mimics OAuth ➔ Authenticate a network request using tokens ➔ Secure network requests by ensuring the use of HTTPS ➔ Make a HTTP POST request to modify data stored by a remote server
Improving Networking with MVC	➔ Refactor an existing application to separate network functionality into its correct role within the MVC design pattern ➔ Create a usable interface that controllers can use to make network requests
Closures Reloaded	➔ Create closures by assigning functions to a constant or variable ➔ Specify closure (function) types for use as values and parameters ➔ Define functions which accept closure parameters ➔ Use type aliasing to simplify the use of complex types ➔ Define and use functions within functions

GCD and Queues	➔ Define and utilize queues for grouping related processes ➔ Run code asynchronously using Grand Central Dispatch ➔ Avoid common pitfalls by ensuring the use of the main thread for situations involving UIKit and CoreData
Backgrounding Lengthy Tasks	➔ Download large files from the network synchronously. ➔ Download large files from the network asynchronously. ➔ Use completion handlers to update the user interface after a network request.
Suggested Electives	➔ iOS Debugging

Project 5: Virtual Tourist

You will create an app that downloads and stores images from Flickr. The app will allow users to drop pins on a map, as if they were stops on a tour. Users will then be able to download pictures for the location and persist both the pictures, and the association of the pictures with the pin.

Supporting Lesson Content: Data Persistence

Lesson Title	Learning Outcomes
Simple Persistence	➔ Learn about simple persistence and how to save small pieces of data. ➔ How to set user preferences, using NSUserDefaults. ➔ Practice setting simple preferences to an existing app.
iOS File System and Sandboxing	➔ Learn about the iOS File System, the “sandbox”. ➔ See how to access these files using NSFileManager. ➔ Use the file manager to save and read a file.
Introducing Core Data	➔ Meet Core Data, Apple’s framework for managing the data layer. ➔ Explore what a data layer is. ➔ Convert a non-Core Data note-taking app to have a Core Data model.
The Core Data Stack	➔ Set up the classes we need to get Core Data up and running. ➔ Use the stack to manage model object creation and deletion. ➔ Persist changes so that data stays put when you restart the app or device.
Simpler Code with Core Data	➔ Enable user interfaces to reactively update whenever the model changes. ➔ Set up an NSFetchedResultsController to observe data changes and notify the UI. ➔ Modify a table view to work with a fetched results controller as its data source. ➔ Turn on caching to reduce how often apps ask the store for data.

Rounding Out Core Data	➔ Update the data model and safely migrate user data between versions. ➔ Work with multiple managed object contexts for different types of tasks. ➔ Keep the user interface responsive by sending lengthy tasks to a background queue.
Selective Electives	➔ Objective-C for Swift Developers ◆ Project 0-C: Interoperability Problem Set (Optional) ➔ Firebase in a Weekend ➔ Firebase Analytics

Project 6: You Decide!

This is your chance to let your iOS Developer skills shine! For this final project, you'll design your own iOS app, taking the design from drawing board to App Store.

Supporting Lesson Content: Final Project

Lesson Title	Learning Outcomes
Research	➔ Brainstorm app ideas and decide on an app and feature list that is realistic and exciting ➔ Sketch UI storyboards and outline expected app use cases and flows ➔ Research and experiment with APIs, web services, and libraries that could be useful for an app idea
Build	➔ Adhere to a proven development process to create quality iPhone and iPad apps. ➔ Build an app and collect user feedback ➔ Fix crashes and bugs to improve the quality of an app
Reflect	➔ Reflect on development, what has been learned, and what should change for future development ➔ Monitor App Store feedback
Selective Electives	➔ Technical Interview Prep ➔ Mobile Design Patterns