ATLS 4120: Mobile Application Development
Week 1: Intro to iOS App Development

SDK Components
- Xcode
  - Integrated Development Environment (IDE) to create and manage development projects
  - Interface Builder
    - Tool to build your application user interfaces
- Frameworks
  - Software libraries that provide specific functionality
- Simulator
  - Simulates running your apps on your Mac
  - Enables you to run, test, and debug iPhone and iPad apps on your Mac
  - Recreates most of the physical behavior of an actual device. Except:
  - No phone, camera, GPS, Accelerometer, iPod, bluetooth
  - Doesn't emulate memory usage
- Instruments
  - Gather and analyze data on your apps behavior

App development flow
- Start with an Xcode template
  - Enables you to easily create different types of apps
  - Single View app
    - Good for apps with a single view
    - Views are the basic building blocks of all UI elements on the iPhone/iPad
    - A view is a rectangular portion of the screen that draws content
- Design the user interface (UI)
- Hook up the UI and the code
- Write the code using the frameworks
- Build and run your app (iteratively)
  - Compiles your files creating a single library
- Test and debug until you're done

Xcode
Use Spotlight to search for Xcode
You should get a welcome screen with a choice to create a new project.
Or menu: File | New | Project

First app
(helloworld)
iOS Application: Single View Application template
Product name: helloworld
Team: None
Organization name: Your name
This will be used in a copyright notice that Xcode automatically creates.
Organization identifier: ATLAS or something else
Xcode will combine the product name and organization identifier to create a unique bundle identifier for your app.
Language: Swift

Leave Use Core Data, Include Unit Tests, and Include UI Tests all unchecked.
Next
Choose a folder for all your iOS projects.
Leave create local git repository unchecked.
Create.
xCode automatically creates the projects and files needed for helloworld.

Xcode Tour
Chapter 1 goes through Xcode in detail as does Xcode help https://help.apple.com/xcode/mac/10.0/

Toolbar
Lets you perform tasks to build and run your project
Left drop down lets you choose how you want to build and run your project
In the middle is the activity view which shows what's going on at any given time

Library
Object library that contains all the UI objects you can use in your user interface by simply dragging and dropping them onto your view.

Editor
- Standard editor-single pane to edit a file
- Assistant editor-2 panes to see related files
- Version editor-compare versions

View lets you show or hide the navigator, debug, and utilities pane.
Organizer shows related information including the documentation.
Jump Bar
Lets you quickly jump to different elements in your project.
- Popup menu
- Forward/back files
- Segmented popup arranged as the hierarchy of your project

Navigator
Let's you view all the files that make up your project.
Organized in Xcode but it does not relate to where the files are really organized on your hard drive.
Project view:
- Project files – the files that make up your project
- Supporting files – all other files including images
- Frameworks – the iOS libraries your project uses
- Products – the application binary

Source Control
Symbols
Search
Issues – errors or warnings
Test
Debug
Breakpoint
Report

Utility Pane
Provides context-sensitive information.

- File inspector
- Quick help

Interface Builder Tour
Click on the Main.storyboard file to see the view created which right now will be blank.
Interface Builder lets you visually build your interface

- What your layout will look like
- How your interface behaves and interacts

The storyboard defines the interface and relationships of your views
Right now your storyboard has one view. Click it.
You'll see the view has a safe area, this is the area that is always visible and where we'll lay out our view.

Document outline
Click the little button in the lower-left corner of the editing area to bring up the hierarchy view.
This shows the content of the storyboard, split up into scenes.
We have just one scene, called View Controller Scene that has a View Controller, which in turn has a View (along with some other things you'll learn about later).
View is the object with the area that a user can see and interact with. It was created when we selected the Single View Application template.
All the user interface objects will go in the view's safe area.
This will be a useful way to see everything in your storyboards.
First Responder is the object that the user is currently interacting with.
Exit will be useful when we have multiple views.
You can also see these by clicking on the top bar of the view.

Utility View
Information for the currently selected objects. This is where you set various properties.
File inspector
Quick Help
Identity inspector
Attributes inspector
Size inspector
Connections inspector

Library
File templates
Code Snippets
Media library
Search/filter field at the bottom of the library.

Building Hello World
Click on Main.storyboard
You'll notice that the default view is for an iPhone Xr. If you click on View as: down below you can change this. You can leave it, doesn't matter for now.
Click on the Object library (square with a circle)
Scroll or search for label

Drag a Label from the library into the View window.
Put it anywhere in the safe area. You might see dotted blue lines that help center it.
Double click the label and type Hello World! in it.
You can use the handles to stretch the label to see the whole text.
Up in the toolbar you will see the name of your project. If you click on it you can chose a simulator.
Pick iPhone Xr (or other, it doesn't really matter)
Build & Run to compile it and run it in the Simulator(ignore warnings for now).
You can play around with your text by going into the attributes inspector and changing its color, font, shadow, background color, etc.

Interactivity
Now let's make it interactive so our app does something.
Now drag a button into the view.
Place it below the label.
Double click the button and add some text like "Say hello"(this changes the title in the button attributes)
Note where the label and button got added in the view hierarchy.
You can run it again in the simulator but you can preview the layout even quicker.
Show the Assistant Editor(middle editor button, 2 circles) and then in the jump bar click on Automatic and choose Preview. This will show the layout on an iPhone. Using the '+' on the bottom left of that pane you can add as many devices as you want to preview.
If you run it in the Simulator you'll see that it doesn't do anything yet.

Xcode created a file called ViewController.swift
This view controller will control our view.
// are comments
`import UIKit` imports the UIKit framework which is the foundation of all iOS UI (import is like include in C)
`class ViewController: UIViewController` defines our class ViewController.
`UIViewController` is the superclass (which is in UIKit which is why we imported that)
All the code for this class will be in the curly braces.

Now we have to connect the interface and the code.
Go back into Main.storyboard.
We also need to see the swift file so open up the assistant editor. Go to Automatic if your Swift file doesn't open.
Click on the button in your storyboard and then hold down the control key.
Then click and drag from the button over to the swift file.
Notice the blue fishing line being drawn between these two. This is how we'll connect them.
Move your cursor between the curly braces for the class. It doesn't matter if it's above or below the boilerplate methods.
When you see a grey box appear release the mouse button.
This window lets you set up the connection between the button and your code.
Connection: Action
Name: buttonPressed
Type: UIButton
Event: Touch Up Inside is the standard event to use for buttons.
Arguments: Sender
Now hit Connect.
You should now see in the swift file

```
  @IBAction func buttonPressed(_ sender: UIButton) { }
```
This is a method called buttonPressed that will be called when the user taps the button.

Before we implement buttonPressed we have to connect our label.
Control-click from the label and drag over to the swift file.
Connection: Outlet
Name: messageText
Type:UILabel
Leave storage as weak.
Connect

This created a variable in the swift file
```
@IBOutlet weak var messageText: UILabel!
```

Make the label a bit larger than its minimum for the text. Now remove the text so it's blank (change it in attributes or double click and delete the text) as we'll be changing the text programmatically.
To still see the outline of the label Editor | Canvas | Show Bounds Rectangles

If you named either of these differently, DON'T change them, it will break it, just go with what you have for now.

Now we're ready to implement the method for the button. Go into the swift file
```
    @IBAction func buttonPressed(sender: UIButton) {
        messageText.text="Hello World!"
    }
```

This assigns the string "Hello World" as the text of label when that action occurs (the user clicks the button).
Notice the autocomplete. Autocomplete is your friend. Get used to hitting return or tab to accept a choice. It makes the typing easier, guards against typos, and helps signal when something is wrong.

Build and Run. Preview is just to see the layout, you must use the Simulator to actually run the app.

<u>Connections</u>
Show the connections inspector and the filled in circles in the swift file for the connections.
Show common connection errors and how to fix them.
Delete the variable messageText(comment out references to it) and run it to show the run time error in the debug error "this class is not key value coding-compliant for the key messageText".
Delete the variable and the connection and make a new connection by creating a new variable first and then dragging from the circle to the image. A filled in circle means it's connected.
```
@IBOutlet weak var msgText: UILabel!
```
Show the connections inspector and that the connection looks the same. Either way works.

<u>App Icon</u>
https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/app-icon/
In the simulator hit the home button to go back to the home screen. (Hardware | Home)
We need an app icon!
In the Project Navigator click on the Assets.xcassets folder.
Click on AppIcon
You'll notice place holders for notification, settings, spotlight, app an d app store for iPhone and iPad.

For now let's just add the app icon for the iPhone app home screen, you can do the others later.
60 pt is points, not pixels. Depending on the resolution there is a different ratio between points and pixels. On early iPhones 1 pt=1 px but On most of the later devices with a Retina display, a single point is actually a 2 × 2 pixel square. On the Plus phones 1 point is a 3 × 3–pixel square.
We will need a 120x120 pixel image for the 2x icon(60x2=120), and a 180x180 pixel image for the 3x icon(60x3=180)
They MUST be png files and be exactly the right size.
PNG is the best format to use as Xcode optimizes them to make them the most efficient to use in iOS apps
iOS will automatically add a mask to make it looks like the others.
Drag the 120x120 png file into the app 2x space.
Drag the 180x180 png file into the app 3x space.
Now run it and go to the home screen(Hardware | Home) to see your app icon!
You will see warnings that you're missing iPad app icons. You can add these or change your app from Universal to iPhone in your helloworld target Deployment Info.

Launch screen
https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/launch-screen/
A launch screen appears instantly when your app starts and is quickly replaced with the first view of your app.
Use the LaunchScreen.storyboard for your launch screen.
Click on LaunchScreen.storyboard and for now just add a label with some text and center it both horizontally and vertically. We'll learn how to use images next week.