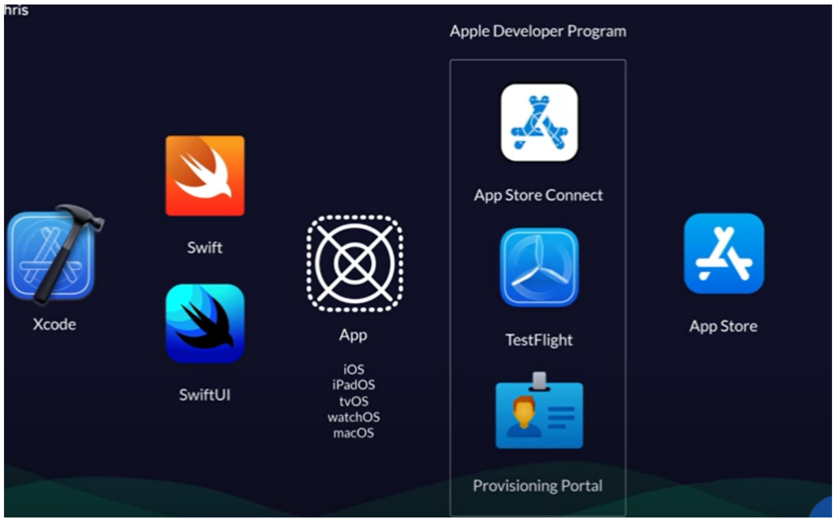
Tutorial notes: https://learn.codewithchris.com/courses/start

Lesson 1:

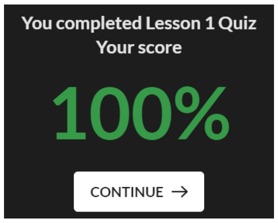
- SwiftUI (a UI framework) can be used to build apps on the ios environment.
- Our IDE: Xcode -> we write the code here.
- The programming language is Swift.
- You can distribute your app to the app store after building your app if you wish to. To do this, you need to join the Apple Developer Program
- TestFlight: used to test, and debug the app (used prior to launch).

Lifecycle of an app, when using Swift:



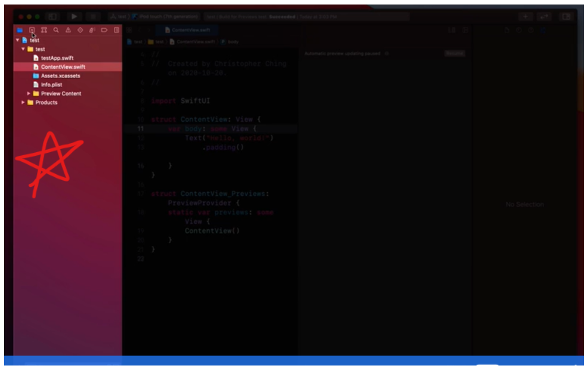
Process (steps):
1. Install Xcode (installing this will automatically install Swift and SwiftUI)
2. (Optional) Join Apple Developer Program if you wish to publish it to the app store)

Quiz 1:


You completed Lesson 1 Quiz
Your score
100%

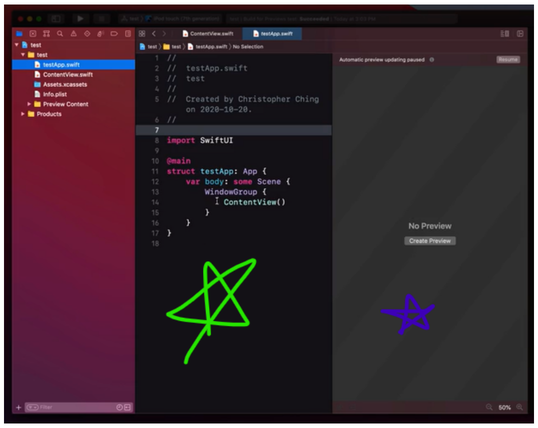
Lesson 2:

Xcode:
Launch mac app store and download Xcode (can also download from https://developer.apple.com/xcode)
Potential roadblock: I am on a Windows PC, which makes the process to set up a lot more complicated. I will try to solve it soon!
Functionality of Xcode:



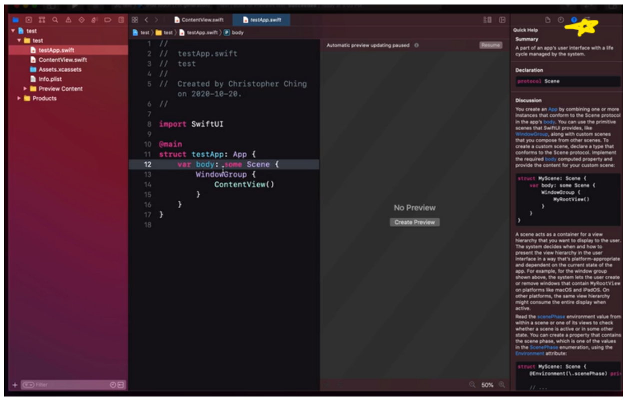
```
//
//  Created by Christopher Ching
    on 2020-10-20.
//

import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
            .padding()

    }
}

struct ContentView_Previews:
    PreviewProvider {
    static var previews: some
        View {
        ContentView()
    }
}
```

The area above marked by the star is called the Navigator Panel: has multiple tabs to go through and switch between files.
Clicking on a file from the Navigation Panel leads you to ->



```
//
//  testApp.swift
//  test
//
//  Created by Christopher Ching
    on 2020-10-20.
//

import SwiftUI

@main
struct testApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

The area above marked in the green star is the Editor area: where we can edit the code from the files displayed in the navigation panel.
The area above marked in the blue star is where you can view a preview



```swift
//
//  testApp.swift
//  test
//
//  Created by Christopher Ching
    on 2020-10-20.
//

import SwiftUI

@main
struct testApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Summary

A part of an app's user interface with a life cycle managed by the system.

Declaration

```swift
protocol Scene
```

Discussion

You create an App by combining one or more instances that conform to the Scene protocol in the app's body. You can use the primitive scenes that SwiftUI provides, like WindowGroup, along with custom scenes that you compose from other scenes. To create a custom scene, declare a type that conforms to the Scene protocol. Implement the required body computed property and provide the content for your custom scene:

```swift
struct MyScene: Scene {
    var body: some Scene {
        WindowGroup {
            MyRootView()
        }
    }
}
```

A scene acts as a container for a view hierarchy that you want to display to the user. The system decides when and how to present the view hierarchy in the user interface in a way that's platform-appropriate and dependent on the current state of the app. For example, for the window group shown above, the system lets the user create or remove windows that contain MyRootView on platforms like macOS and iPadOS. On other platforms, the same view hierarchy might consume the entire display when active.

Read the scenePhase environment value from within a scene or one of its views to check whether a scene is active or in some other state. You can create a property that contains the scene phase, which is one of the values in the ScenePhase enumeration, using the Environment attribute:

```swift
struct MyScene: Scene {
    @Environment(\.scenePhase) pri

    // ...
```

Lastly, the area above marked with the yellow star is the Inspector Area: can be used to configure or edit certain specific parts from the editor area.

To comment: //THIS IS A COMMENT (just like in Java)

Bundle Identifier: the identifier is composed of the product name and organization identifier. These are specified by you when you create a new Xcode project. The identifier's goal is to uniquely identify your app in the App Store. This can be changed at any time if you wish to.

Quiz 2:

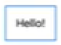
You completed Lesson 2 Quiz
Your score
100%

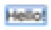
Lesson 3:

You can preview your changes to the code without having to sync or anything, since it auto-syncs.
To generate a label, the syntax looks as follows:

```swift
Text("Hello World").padding()
```

This prints the words "Hello World." The point of the .padding() is so that the frame around doesn't seem too crowded around the word and there is enough spacing out. Here is the difference between using .padding() and not using it:

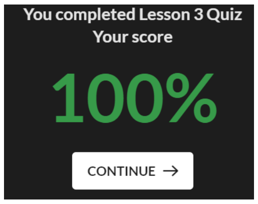
With .padding(): 

Without .padding(): 

See how the first one has more spacing between the frame and the word.
Modifier: is attached to an element to change the way it looks. Example: .padding(), background(Color.color_name)

You can preview the result of your code by either using the preview area (marked by the blue star on the last page), or by running your app
You don't have to write the code, instead you can use the pre-filled templates if needed (from the Library panel). You can add modifiers through the Instructors panel without having to write code, if you wish to.

Quiz 3:


You completed Lesson 3 Quiz
Your score
100%

Challenge question code:

```
Text("Hello world").padding().background(Color.green)
```

The above line does the following:
The red highlighted area prints "Hello World"
The pink highlighted area forms a spacing between the words and the frame around it
The green highlighted area makes the background green (moreso like a green box around the words *Hello World*)

Lesson 4:

You can insert an image using:

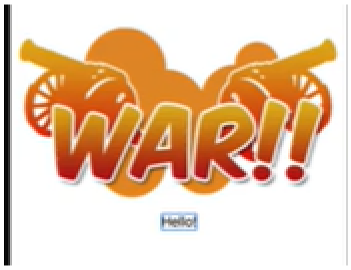
```
      Image("Image name")
```

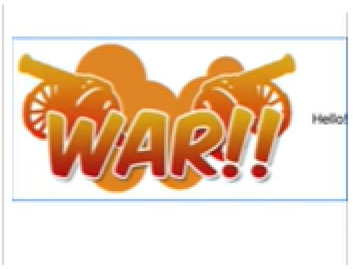
.resizable() is a modifier to change the size of the image
.aspectRation() can zoom in/out

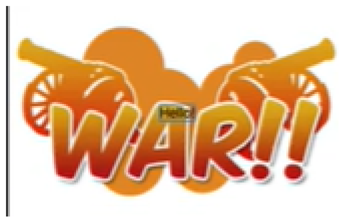
The process for inserting an image is to first add the image to the asset library and use the name assigned to that as the "Image name" in the above line of code.

VStack: vertical stack (you can put upto 10 elements inside a vertical stack, and they can be of varying type). This stacks elements in vertical order (one below the other). The same thing can be done to stack to the right using HStack (horizontal stack). ZStack stacks them on top of each other.
Example:

VStack HStack ZStack







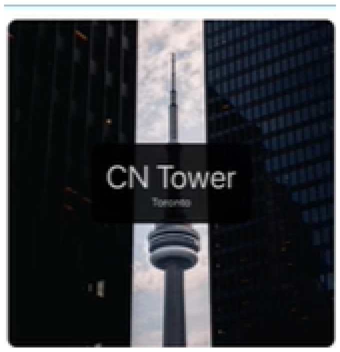
Spacer() creates a space between elements by taking up all the available space within the container.

Quiz 4:

You completed Lesson 4 Quiz
Your score

67%

Mistake: ZStack is a container element (HStack and VStack are as well), and these elements can hold upto 10 other elements. These can be used as ways to represent the different elements and their placements in general. Spacer and Image are elements that can be placed within the containers, but are NOT containers themselves.
Challenge question goal: generate this image:



Code:

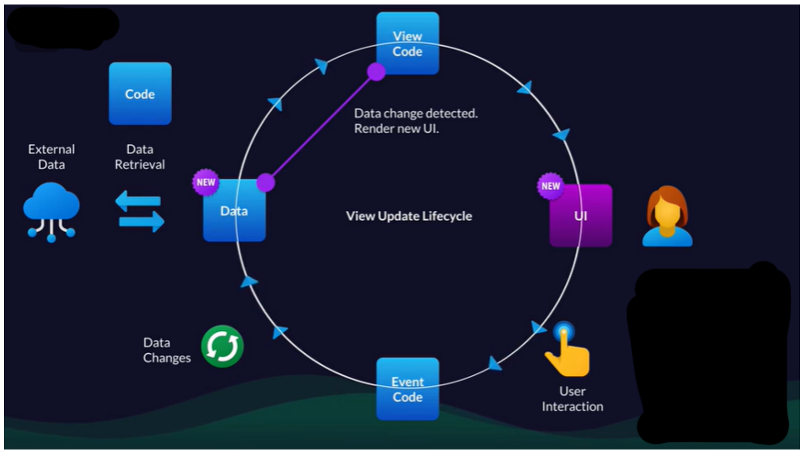
```
ZStack {
    Image("tower").resizable().aspectRatio(contentMode:.fit)
    VStack {
          Text("CN Tower")
             .font(.largeTitle)
             .padding()
             .background(Color.black)
          Text("Toronto")
    }
}
```

Here, the VStack block stacks *CN Tower* and *Toronto* such that *Toronto* is below *CN Tower*. The font size of *CN Tower* will be huge due to the .font(.largeTitle), and there will be padding to add space around the word. Then *Toronto* is added below *CN Tower*. Then, this block is stacked on top of the image of the tower, using the ZStack block.

Lesson 5: DEMO (no notes for this section)

Lesson 6:

The lifecycle of an app and how it is updated:



These are the steps followed:
Any data that is required for the app is accessed and retrieved, and then the UI interface that you built shows up according to that data (if there is no data linked to you app, then the UI interface shows up directly, without having to do much with the data). When the user interacts with the app, like say create an account by typing in an email, password, and clicking the "GO" button, then this data is saved in the existing data and the app's UI is "refreshed" to represent and adjust to the updated data. This keeps repeating on and on until the end (or when you quit the app).

What is the data in this case?
Just like in Java, there are the following data types in Swift:
1. String: a chain of characters. Example: "SwiftUI is fun!"
2. Int: a non-decimal number. Examples: 5, 8, 100
3. Double: a decimal number. Examples: 8.64545, 23.0
4. Bool: a true or false value

Declaring a variable in Swift:

```
var name:Type = value_in_var
```

Example

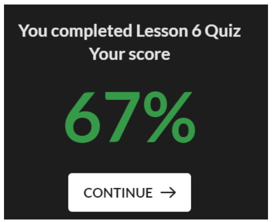
```
var className:String = "VIP"
```

```
var numberOfPeople:Int = 10
```

Just like in Java, you cannot switch the type of a variable after declaration. You can however set it to a different value of the same type if you wish to, unless the variable is a constant.
Declaring a constant in Swift:

```
let className:String = "Best VIP ever!"
```

Quiz 6:


You completed Lesson 6 Quiz
Your score
67%

Mistake: If you don't specify the data type at the time of declaration, then Xcode infers it's type based off the first piece of data you assign to the variable

Lesson 7:

Function syntax:

```
func name(parName:Type) { //defines the function
      //body of function
}

name() //calls the function
```

Example

```
func add(a:Int, b:Int) {
      print(a + b)
}
add(a: 55, b:  45)
```

This prints 100

A <u>function</u> is basically a block of code that can be executed by just calling the name of the function rather than repeatedly typing out the code itself. This comes in handy when doing repetitive tasks, since you can just create the function once and keep calling it again and again, how many ever times you want. A function is the same thing as a method in Java (it is called a function in Python as well).

A function that doesn't return anything has type Void (which you don't have to explicitly state)
A key thing to note here is that the scope of a variable is only within the set of curly braces it is in, so if you declare a variable in a function, you cannot access that variable outside that function. A good way to keep that value might be to return it, or set it to a global variable which has a larger scope and can be accessed out of the function as well.

Quiz 7:

You completed Lesson 7 Quiz
Your score
100%

Challenge 1: The function should print out "Good Morning" in the console

```
func goodMorning() {
      print("Good Morning")
}
```

Challenge 2: Calculate tax on subtotal (13% tax), and return it. Then call it on 100

```
func getTotalWithTax(subtotal: Double) {
      var num:Double = subtotal * 1.13
      return num
}
var sub:Double = 100.0
print(getTotalWithTax(sub))
```

Here, we are creating the function first, where subtotal is a parameter of type Double, then we are calculating the tax and returning it. Lastly, we are printing the result of calling the function on 100, so we are calculating the tax for $100 and returning the with tax total

Lesson 8:

A Structure (or struct), can hold multiple variables, constants and functions (it can be viewed similar to a class)
This is what a struct looks like:

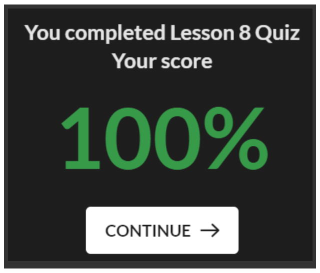
```
struct nameStruct {
func goodMorning() {
            print("Good Morning")
      }
func getTotalWithTax(subtotal: Double) {
            var num:Double = subtotal * 1.13
            return num
      }
      var sub:Double = 100.0
      print(getTotalWithTax(sub))
}
```

You can use a struct to create entities that communicate with each other to build a bigger program. For instance, you could use a struct to build a chatbot using a struct, where each function such as reading the message, responding to the message, etc. can be carried out by a separate function within the same struct , and there can be variables to hold any important data to be accessed at any point within the struct.

A struct is a great way to organize your code by functionality, to represent each functionality in your app with a specific block of code rather than being all over the place.

In Swift, functions and methods are slightly different. A function can be any block of executable code written as a block to be called on, while a method is a function that is in a struct
Property: A property is used to refer to a variable that is inside a struct

Quiz 8:


You completed Lesson 8 Quiz
Your score
100%

Lesson 9:

Instance: you can create an instance of a structure by using the struct that you declared. A good way to understand this is to think of the declaration of the struct as a blueprint to a house, and the instance of calling it is the house that you are building using the blueprint (it is a specific instance of that struct). The instance's datatype is that of the struct. You can create an instance of that data type, and then you can access and set variables in the struct through the name of the instance of the struct.

Example:

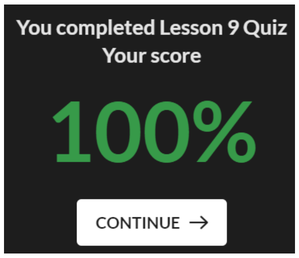
```
struct happy() {
      var feeling = "very glad"
      func currentState() {
            print(feeling)
}
}
var state:happy = happy() //this created an instance of happy (the data type here is happy)
state.feeling = "a bit sad" //this sets the variable feeling for this instance "state" to be a bit sad
state.currentState() //this prints a bit sad since that is what we set the variable feeling equal to for this instance of happy (which is called state)
```

So basically, the only way you can use a struct effectively is by creating an instance of it

A key thing to note is that when you are accessing the attributes of an instance of a struct, you have to use the dot-notation, not on the name of the struct, but on the name of the instance of the struct that you created.

Lastly, you can create however many instances of a struct that you want to and all of them are independent, sso if you set a variable to a certain value, for a certain instance of the struct, then other instances of that struct are not impacted or changed, and nor is the original value of that variable declared in the struct. For instance, changing feeling to "a bit sad" above only changed the value of the variable feeling for the instance state of the struct happy, not the value of the variable in the struct happy itself. So if you create a new instance of the struct happy and printed out the value of the variable feeling for the new instance (without changing it), then it will print "very happy" as designated in the declaration of the struct.

Quiz 9:


You completed Lesson 9 Quiz
Your score
100%

Challenge:
Declare a struct called TaxCalculator
Declare a property inside called tax and set it to a decimal value representing the amount of sales tax where you live
Declare a method inside called totalWithTax that accepts a Double as an input parameter and returns a Double value.
Inside that method, write the code to return a Double value representing the input number with tax included
Create an instance of the struct and print the total when the subtotal is 180, in Texas, where sales tax is 8%

```swift
struct TaxCalculator() {
      var tax:Double = 6
      func totalWithTax(subtotal: Double) -> Double {
            var total:Double = (1+(tax/100)) *  subtotal
            return total
      }
}

var calc:TaxCalculator = TaxCalculator()
calc.tax = 8
print(calc.totalWithTax(180))
```

Lesson 10:

Buttons are an important part of the UI for most apps, since they are a very common way for users to interact with the app.

How do you build a button using SwiftUI?

```
Button("Text on button", action: () -> what happens when you click the
button?)
```

Example:

```
Button("Click Me", action: {
     print("Hey!")
} )
```

So this basically creates a button with the text "Click Me" on it, that prints "Hey!" to the console when you click on it

You can use any containers (such as VStack, HStack, or ZStack with buttons, so that you can position buttons as you wish along with a combination of text, images, etc.).

The two important parts of a button creation are the label and action. The label is what text is on the button and the action is what happens when you click the button. The action is very important since this is what makes the app interactive and helps users change/update the data that is a key part of app functionality.

Closure: A closure is a block of code that is enclosed by curly braces and it's very similar to a function, but it doesn;t have a name. The way we declare the action for a button is using a closure, because we are simply telling it what to do in curly braces, rather than giving it a function per se which is labeled by a name for the function. Closures are passed into methods and functions just like parameters.

To be specific, the action is a trailing closure which isn;t in the parameter list, but is specified at the end of the method call. If you see you can tell that this is the case for actions

Quiz 10:

You completed Lesson 10 Quiz
Your score
100%

Challenge (stack an image and a button. The button should print something to the console):

```
HStack {
      Image("tower").resizable().aspectRatio(contentMode:.fit)
      Spacer()
Button("CN Tower", action: {
          print("We are happy that you want to learn more about the CN
Tower. To do so, visit cntower.com")
      })
}
```

Lesson 11:

How do you change data when users interact with the app?
State property: A state property is a property that the view code relies on to render its UI. The way you add it is by including @State to a property at the beginning of its declaration. This will denote that this property is used to display the UI. State properties will enable changing the data that they store . This is very important, because app functionality hugely relies on data updating properly based on user interactions and state properties are key for that. The state property is a wrapper (very similar to a wrapper class in Java!!).
In your view code, all you have to do is reference your properties by their name. The property name is enough to sort of "form a link" here between the data and the property
Quiz 11:

You completed Lesson 11 Quiz
Your score
100%

Lesson 12:

Conditionals are a very important part of any programming language, since they open up the scope of what you can do with the language. A conditional is a statement that says if this is true, do this, otherwise, do that. The syntax of an if statement is as follows in Swift.
&& is and
|| is or
== is to check if something is equal to something else (vs. = used to set the value of a variable)
!= checks if two things are not equal
> greater than
>= greater than or equal to

```
if (true && false) {
      print("This won't print, because false and anything is always false")
} else if (false || true) {
      print("This will print since true or anything is true")
} else {
      print("We won't reach this statement")
}
```

This is a very trivial example, but you can check the truth of things that are a lot more complicated when you use if statements. An important thing to notice is that we kick out of the statement as soon as you hit an if or else if that is true. You reach else only if all the if/else if statements are false. This is why the else statement is never reached in the above block of code.

You can also have other functionality like calling a function if the condition is true, or create an instance of a struct if a statement is false, etc.

Quiz 12:

You completed Lesson 12 Quiz
Your score

100%

End of tutorial!