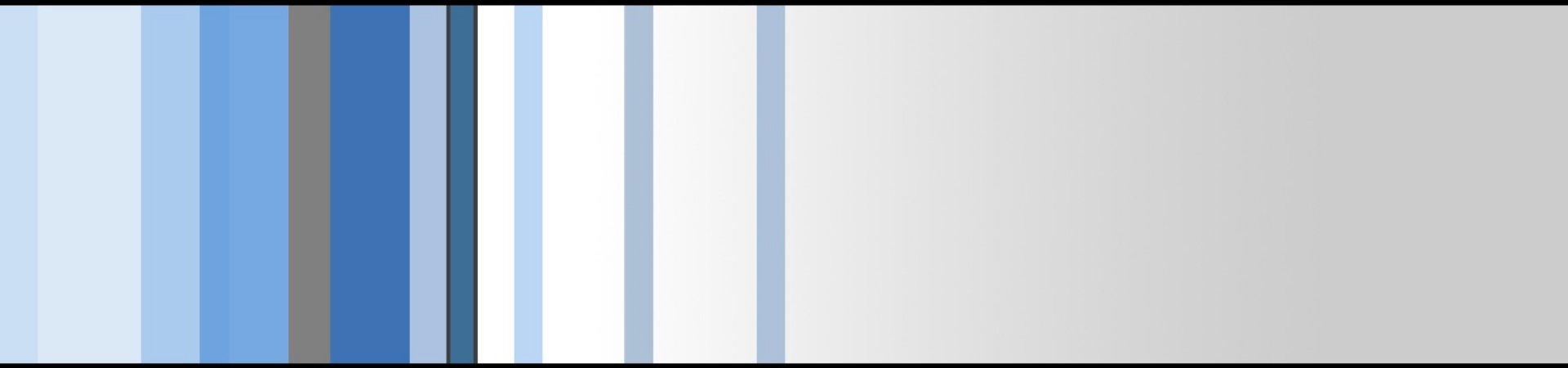
Class 2

CSC 496: iOS App Development
Swift Fundamentals: Operators, Control Flow and Xcode

Si Chen (schen@wcupa.edu)



Constants, Variables, and Data Types

- Constants and variables must be declared before they're used. You declare constants with the **let** keyword and variables with the **var** keyword.

```
let maximumNumberOfLoginAttempts = 10
var currentLoginAttempt = 0
```

- You can declare multiple constants or multiple variables on a single line, separated by commas:

```
var x = 0.0, y = 0.0, z = 0.0
```

Type Annotations

- You can provide a ***type*** *annotation* when you declare a constant or variable, to be clear about the kind of values the constant or variable can store.
- Write a type annotation by placing a colon after the constant or variable name, followed by a space, followed by the name of the type to use.

```
var welcomeMessage: String
```

Printing Constants and Variables

- Swift uses *string interpolation* to include the name of a constant or variable as a placeholder in a longer string, and to prompt Swift to replace it with the current value of that constant or variable.
- ***Wrap the name in parentheses and escape it with a backslash before the opening parenthesis:***

```
print("The current value of friendlyWelcome is \(friendlyWelcome)")
// Prints "The current value of friendlyWelcome is Bonjour!"
```

Semicolons

- Unlike many other languages, Swift **doesn't require** you to write a semicolon (;) after each statement in your code, although you can do so if you wish.
- However, semicolons *are* required if you want to write multiple separate statements on a single line:

```
let cat = "🐱"; print(cat)
// Prints "🐱"
```

Most Common Types in Swift

Name	Type	Purpose	Example
Integer	`Int`	Represents whole numbers, or integers	`4`
Double	`Double`	Represents numbers requiring decimal points, or real numbers	`13.45`
Boolean	`Bool`	Represents `true` or `false` values	`true`
String	`String`	Represents text	`"Once upon a time..."`

Type Inference

- You may have noticed that you **don't have to specify the type of value when you declare a constant or variable**. This is called **type inference**. Swift uses type inference to make assumptions about the type based on the value assigned to the constant or variable.

```
let cityName = "San Francisco"
// "San Francisco" is obviously a `String`, so the compiler
automatically assigns the type of cityName to a `String`.

let pi = 3.1415927
// 3.1415927 is a number with decimal points, so the compiler
automatically assigns the type `pi` to a `Double`.
```

```
let cityName: String = "San Francisco"

let pi: Double = 3.1415927
```

Create your own type

```
1  import UIKit
2
3  struct Car{
4      var make:String
5      var model:String
6      var year:Int
7  }
8
9  var c = Car(make: "BMW",
       model: "X3", year:
       2022)
10 print(c.make)
```

Operators

- Operators are the symbols that make your code work. You'll use them to perform actions like check, change, or combine values.
  - Assign a Value

```
let favoritePerson = "Luke"
```

  - Basic Arithmetic

```
var opponentScore = 3 * 8
// opponentScore has a value of 24
var myScore = 100 / 4
// myScore has a value of 25
```

Operators

- Compound Assignment

```
myScore += 3 // Adds 3 to myScore
myScore -= 5 // Subtracts 5 from myScore
myScore *= 2 // Multiples myScore by 2
myScore /= 2 // Divides myScore by 2
```

- Numeric Type Conversion

```
let x = 3
let y = 0.1415927
let pi = x + y
```

```
let x = 3
let y = 0.1415927
let pi = Double(x) + y
```

In the revised code, Double(x) creates a new Double value from the Int value x, enabling the compiler to add it to y and assign the result to pi.

Control Flow

- Logical and Comparison Operators

Operator	Type	Description
==	Comparison	Two items must be equal.
!=	Comparison	The values must not be equal to each other.
>	Comparison	Value on the left must be greater than the value on the right.
>=	Comparison	Value on the left must be greater than or equal to the value on the right.
<	Comparison	Value on the left must be less than the value on the right.
<=	Comparison	Value on the left must be less than or equal to the value on the right.
&&	Logical	AND—The conditional statement on the left *and* right must be `true`.
||	Logical	OR—The conditional statement on the left *or* right must be `true`.
!	Logical	NOT—Returns the logical opposite of the conditional statement immediately following the operator

if Statements

```
let temperature = 100
if temperature >= 100 {
  print("The water is boiling.")
}
```

Console Output:

```
The water is boiling.
```

If-else if-else Statements

```
var finishPosition = 2

if finishPosition == 1 {
  print("Congratulations, you won the gold medal!")
} else if finishPosition == 2 {
  print("You came in second place, you won a silver medal!")
} else {
  print("You did not win a gold or silver medal.")
}
```

Boolean Values

```
var isSnowing = false

if !isSnowing {
  print("It is not snowing.")
}
```

Console Output:

```
It is not snowing.
```

Switch Statement

```
let numberOfWheels = 2
switch numberOfWheels {
case 0:
    print("Missing something?")
case 1:
    print("Unicycle")
case 2:
    print("Bicycle")
case 3:
    print("Tricycle")
case 4:
    print("Quadcycle")
default:
    print("That's a lot of wheels!")
}
```

Switch Statement

- Any given case statement can also evaluate multiple conditions at once.

```
let character = "z"

switch character {
case "a", "e", "i", "o", "u":
    print("This character is a vowel.")
default:
    print("This character is a consonant.")
}
```

Switch Statement

- When working with numbers, you can use interval matching to check for inclusion in a range

```
switch distance {
case 0...9:
    print("Your destination is close.")
case 10...99:
    print("Your destination is a medium distance from here.")
case 100...999:
    print("Your destination is far from here.")
default:
    print("Are you sure you want to travel this far?")
}
```

Exercise 1: Grade Calculation Using if-else

- **Problem Description:**

Write a Swift program that takes a student's numerical grade (0 to 100) as input and outputs the corresponding letter grade based on the following criteria:

- A: 90-100
- B: 80-89
- C: 70-79
- D: 60-69
- F: Below 60

- **Sample Code Skeleton:**

```
// Your code here
let numericalGrade = 95

// Implement if-else logic here to determine letter grade
```

Exercise 2: Grade Calculation Using switch

- **Problem Description:**

Write a Swift program that takes a student's numerical grade (0 to 100) as input and outputs the corresponding letter grade based on the following criteria:

- A: 90-100
- B: 80-89
- C: 70-79
- D: 60-69
- F: Below 60

- **Sample Code Skeleton:**

```
import Foundation

// Your code here
let numericalGrade = 95

// Implement switch logic here to determine letter grade
```

Ternary Conditional Operator

```
var largest: Int
let a = 15
let b = 4

largest = a > b ? a : b
```

Exercise 3: Evaluate Number Using Ternary Conditional Operator

- **Problem Description:**

Write a Swift program that evaluates if a given integer is even or odd using the Ternary Conditional Operator.

Sample Code Skeleton:

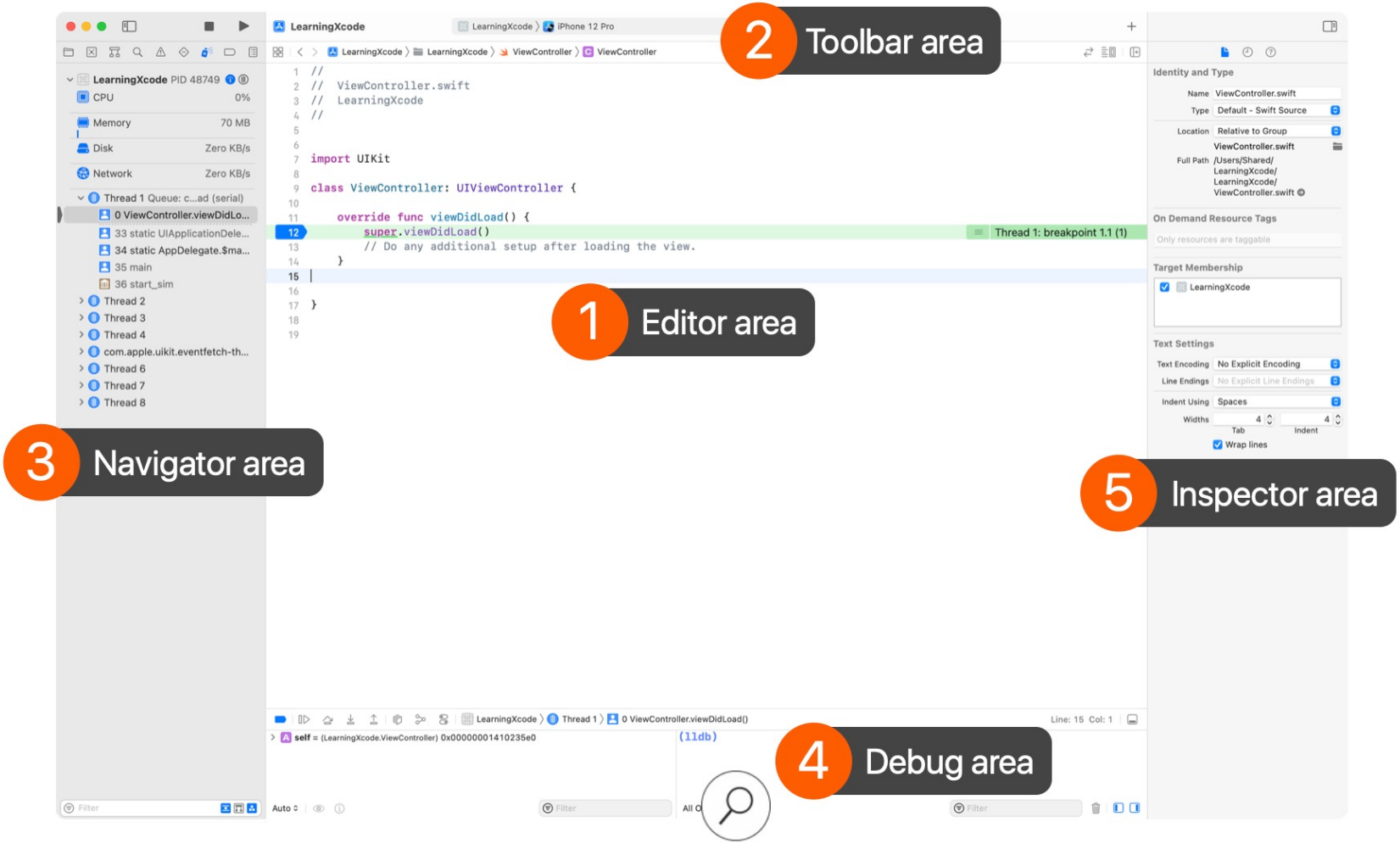
```
import Foundation

// Pre-define an integer
let number = 10

// Use Ternary Conditional Operator to evaluate the number
let result: String = // Your code here

print("The number is \(result).")
```

Please replace the comment *// Your code here* with the appropriate use of the **Ternary Conditional Operator** to check whether the number is even or odd.

Xcode

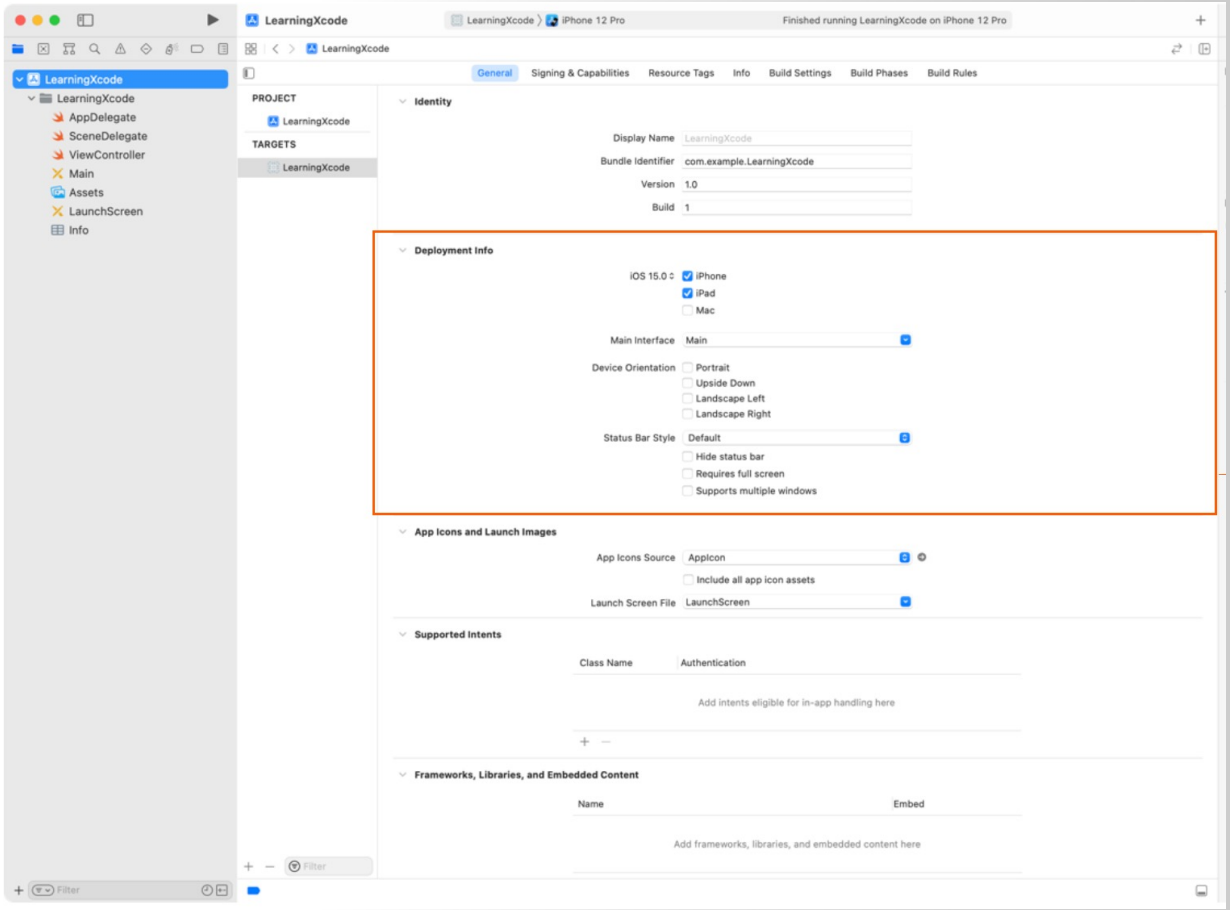
- How to navigate Xcode projects
- How to use the Project navigator, debug area, assistant editor, and version editor

Xcode Interface



.xcodeproj Project file

- .xcodeproj is the project file, which includes all the settings for your project and its targets. Each target is a product that Xcode can build from the project.



Building, Running, and Debugging an App

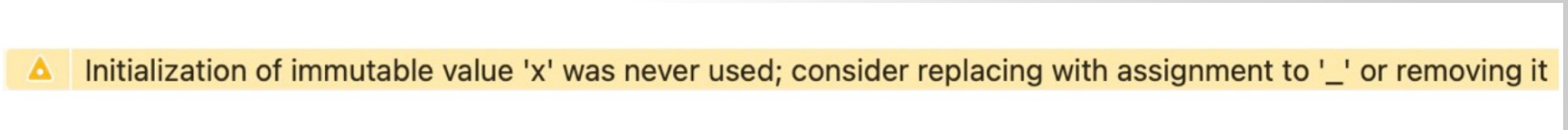
Click the Run button, or use the keyboard shortcut (Command-R) to begin launching the app in Simulator.

Debugging an Application

- When you run an app as described above, either on Simulator or on your device, **Xcode will connect the app to its debugger**. This allows you to watch the execution of your code in real time, stop code execution using breakpoints, print information from your code to the console, and much more.
- As you proceed through this course, you'll encounter three types of issues: **warnings, compiler errors, and bugs**

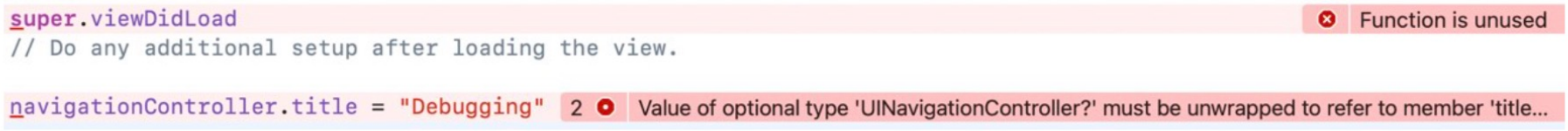
Warnings:

- Writing code that never gets executed
- Creating a variable that never changes
- Using code that's out of date (also known as deprecated code)



Debugging an Application

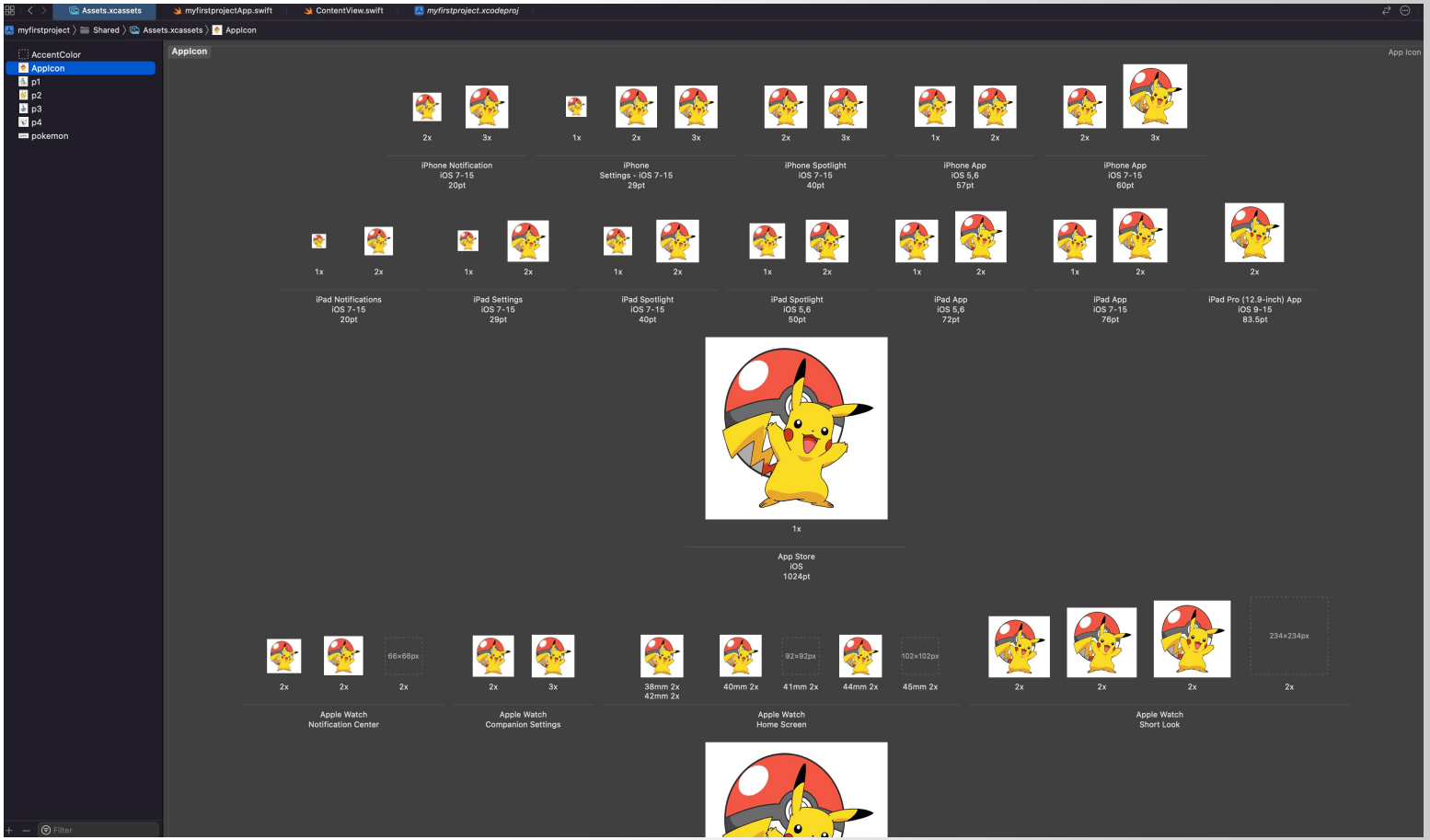
Compiler Errors: an error prevents the code from ever being executed. Simulator won't even launch if your code has an error.



Bugs: the hardest issue to track down. A bug is an error that occurs while running the program, resulting in a crash or incorrect output. Finding bugs can involve some time and some real detective work.

```
var names = ["Tammy", "Cole"]
names.removeFirst()
names.removeFirst()
names.removeFirst()
```

Build the first App



assets.xcassets → AppIcon

generate app icon: https://appicon.co/#app-icon

Create Button

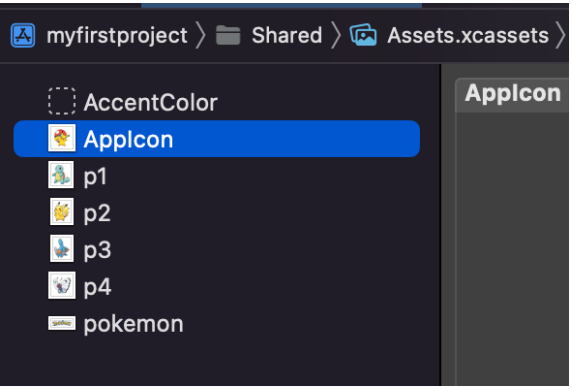
To create a button with a string title you would start with code like this:

```
Button("Button title") {
    print("Button tapped!")
}
```

Create Button

```
8  import SwiftUI
9
10 struct ContentView: View {
11     @State private var showDetails = false
12     var body: some View {
13         Text("Hello, world!")
14             .padding()
15         Button("Show details") {
16             showDetails.toggle()
17         }
18
19         if showDetails{
20             Image("pokemon")
21             Text("Good job!!!").bold()
22         }
23     }
24 }
25
26 struct ContentView_Previews: PreviewProvider {
27     static var previews: some View {
28         ContentView()
29     }
30 }
```

Add Image

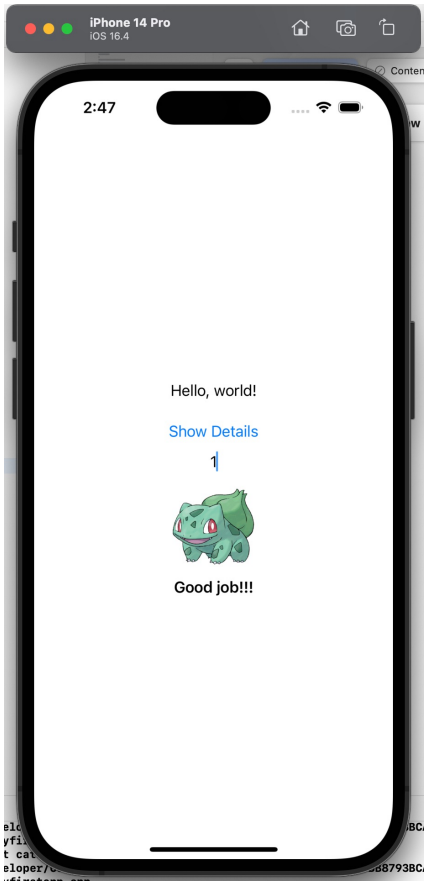


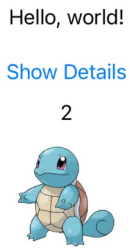

```swift
import SwiftUI

struct ContentView: View {
    @State private var showDetails = false
    var body: some View {
        Text("Hello, world!")
            .padding()
        Button("Show details") {
            showDetails.toggle()
        }

        if showDetails{
            Image("pokemon")
            Text("Good job!!!").bold()
        }
    }
}

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

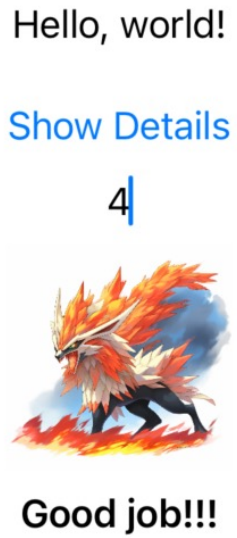
Create TextField

```
8  import SwiftUI
9
10 struct ContentView: View {
11     @State private var showDetails = false
12     @State private var pokemonID = "1"
13     var body: some View {
14         Text("Hello, world!")
15             .padding()
16         Button("Show details") {
17             showDetails.toggle()
18         }
19         TextField("pokemon ID", text: $pokemonID).multilineTextAlignment(.center)
20
21         if showDetails{
22             Image("pokemon")
23             Text("Good job!!!").bold()
24         }
25     }
26 }
27
28 struct ContentView_Previews: PreviewProvider {
29     static var previews: some View {
30         ContentView()
31     }
32 }
```

Exercise: Build an App to display Pokemons







Q & A