Data Flow

Introduction to SwiftUI

Hacking Spatial Computing • National University of Singapore

Constants, Variables & Types

State Variables & Buttons

Binding Variables

Conditional Rendering

Constants, Variables & Types

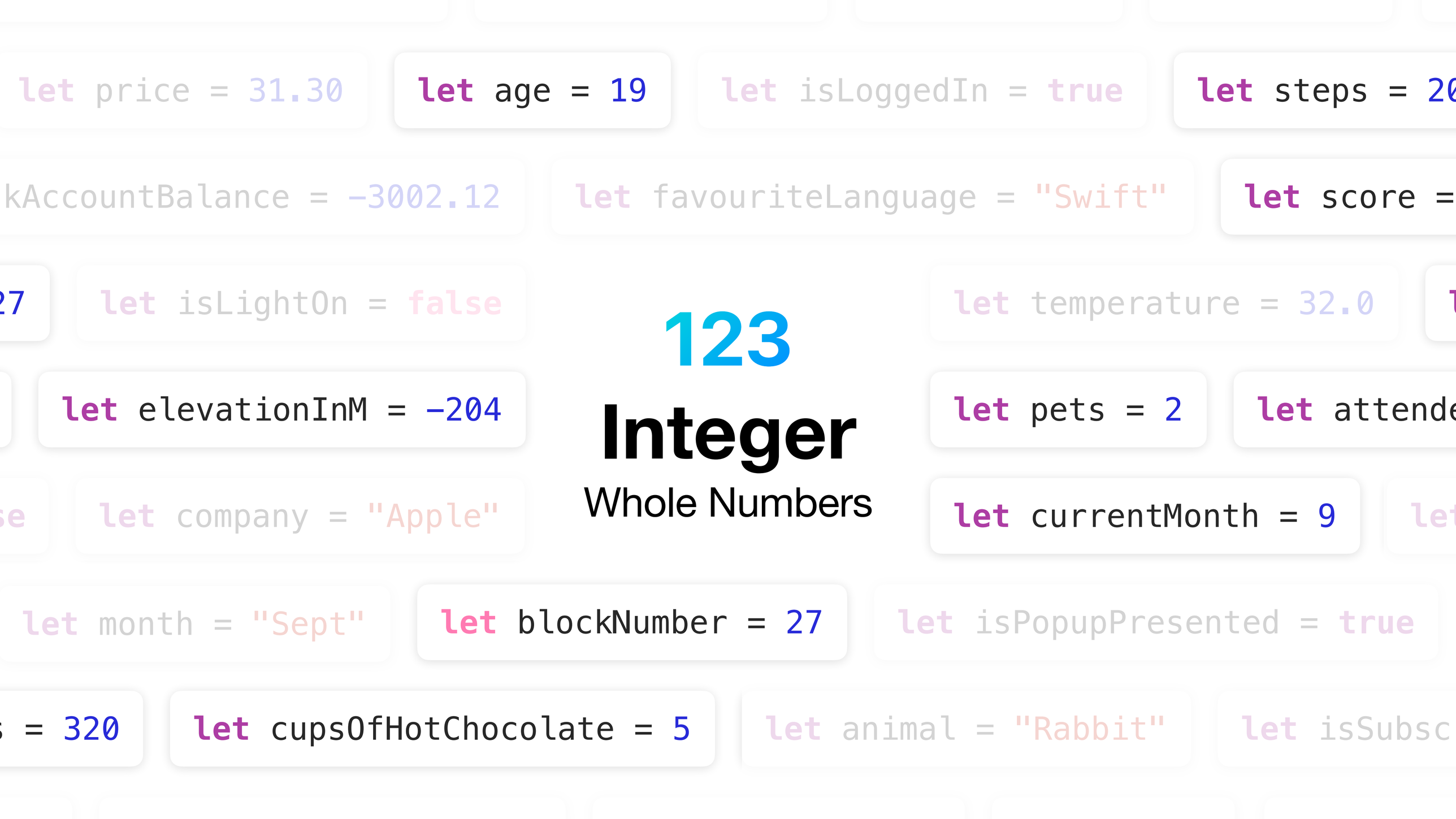
State Variables & Buttons

Binding Variables

Conditional Rendering

Data Types

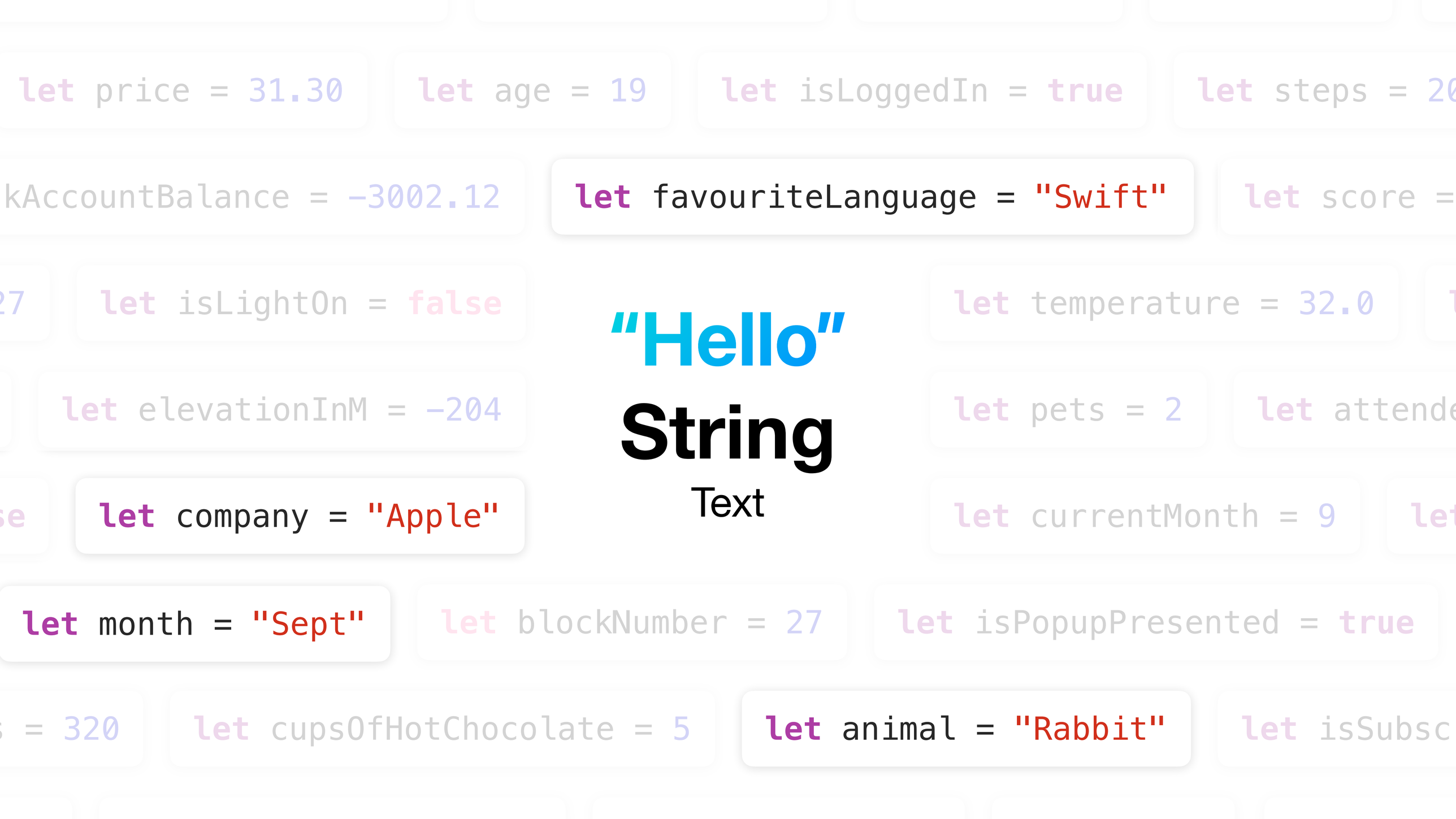
```
let price = 31.30
let age = 19
let isLoggedIn = true
kAccountBalance = -3002.12
let favouriteLanguage = "Swift"
let isLightOn = false
let temperature = 32.0
let elevationInM = -204
let pets = 2
let company = "Apple"
let currentMonth = 9
let month = "Sept"
let blockNumber = 27
let isPopupPresented = true
= 320
let cupsOfHotChocolate = 5
let animal = "Rabbit"
```


let price = 31.30
let age = 19
let isLoggedIn = true
let steps = 20
kAccountBalance = -3002.12
let favouriteLanguage = "Swift"
let score =
27
let isLightOn = false
let temperature = 32.0
123
Integer
Whole Numbers
let elevationInM = -204
let pets = 2
let attende
se
let company = "Apple"
let currentMonth = 9
let month = "Sept"
let blockNumber = 27
let isPopupPresented = true
s = 320
let cupsOfHotChocolate = 5
let animal = "Rabbit"
let isSubsc


let price = 31.30
let age = 19
let isLoggedIn = true
kAccountBalance = -3002.12
let favouriteLanguage = "Swift"
let isLightOn = false
let temperature = 32.0
3.14
Double
Decimals
let elevationInM = -204
let pets = 2
let company = "Apple"
let currentMonth = 9
let month = "Sept"
let blockNumber = 27
let isPopupPresented = true
let cupsOfHotChocolate = 5
let animal = "Rabbit"

Boolean

True/False Values


let price = 31.30
let age = 19
let isLoggedIn = true
let favouriteLanguage = "Swift"
let isLightOn = false
let temperature = 32.0
let elevationInM = -204
let pets = 2
let company = "Apple"
let currentMonth = 9
let month = "Sept"
let blockNumber = 27
let isPopupPresented = true
let cupsOfHotChocolate = 5
let animal = "Rabbit"
"Hello"
String
Text

Variables

A variable is **a data store that can have its value changed whenever you want**.

```
var numberOfCookies = 100

// Tom takes one
numberOfCookies = 99

// Bob drops the jar
numberOfCookies = 0

// The jar starts sucking cookies up???
numberOfCookies = -100
```

Constants

A constant is **a data store that you set once and can never change**.

```
let numberOfCookies = 100

// Tom takes one
numberOfCookies = 99
```

Cannot assign to value: 'numberOfCookies' is a 'let' constant

Change 'let' to 'var' to make it mutable — Apply

Naming

Names can **only include letters, numbers, underscores, and emojis**.

Names **cannot start with a number**.

Names **cannot contain spaces**.

Names **cannot contain reserved words**.

Names **are case sensitive**.

```swift
var personName = "Bob"

var person2 = "Tom"

var _123 = true

var 🍕 = 23

var 🌡️ = 27.5

var 风和日丽 = "早上"
```

```swift
var person name = "Bob"

var 2ndPerson = "Tom"

var 123 = false

var var = 3.14

var $owed = 2004.50
```

Variable and Constants are named in the **camelCaseConvention**. The first letter of the first word is lowercased, the first letter of subsequent words are uppercased.

Type Safety

Swift is a **type-safe language**, which means the language helps you to be clear about the types of values your code can work with.

Type safety helps you catch and fix errors as early as possible in the development process.

```swift
var numberOfCookies = 100

// Tom ate half a cookie
numberOfCookies = 99.5
```

Cannot assign value of type 'Double' to type 'Int'

Replace '99.5' with 'Int(99.5)' Apply

```
var numberOfCookies = 100.0

// Tom ate half a cookie
numberOfCookies = 99.5
```

Type Annotation

By default, Swift infers a data store's type based on its value. Type annotations allow you to **explicitly set a data type**.

```swift
// This is an Integer
var numberOfCookies = 100

// This is now a Double because you say so
var numberOfCookies: Double = 100
```

On Appear

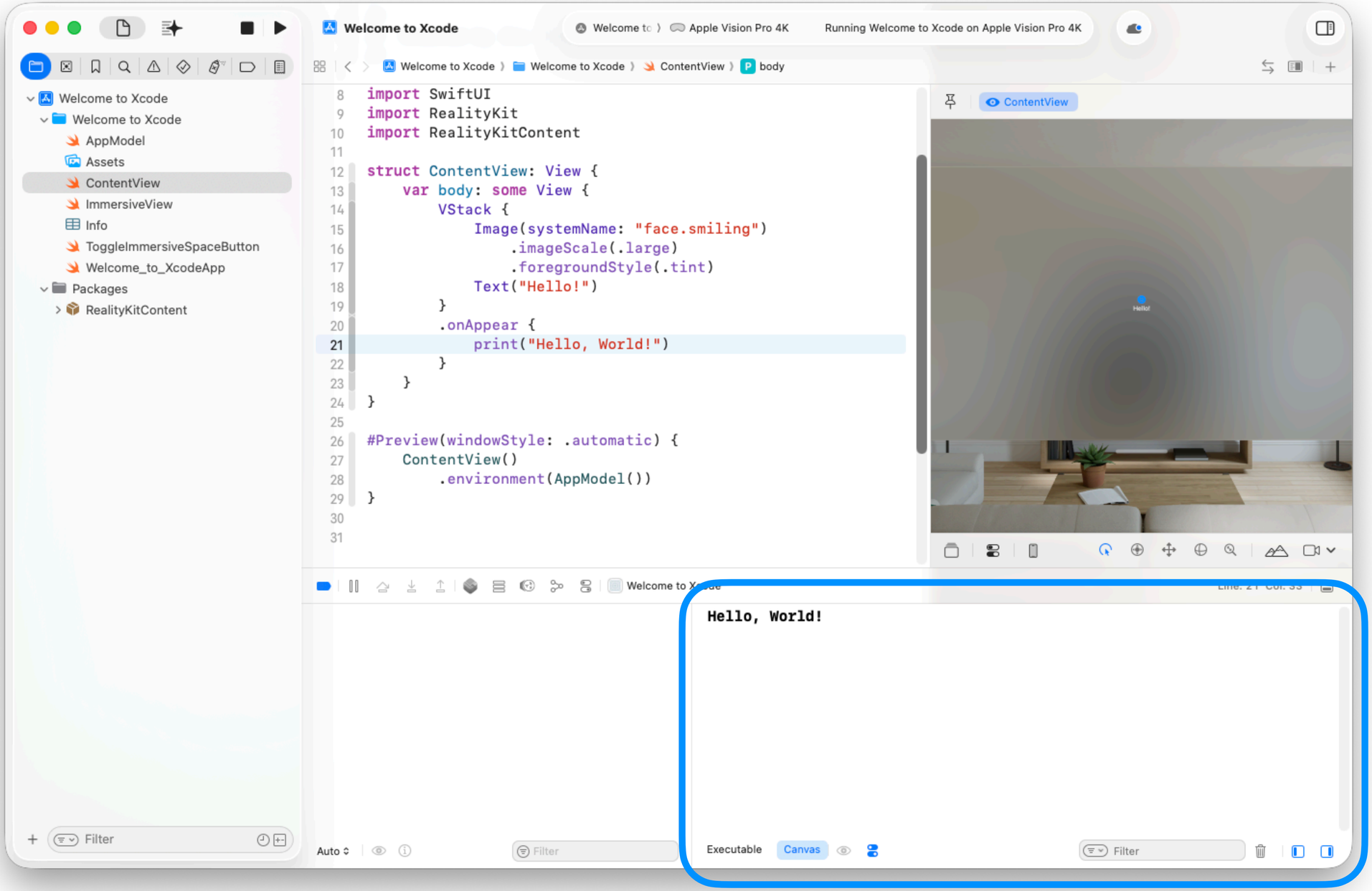
```
struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundStyle(.tint)
            Text("Hello, World!")
        }
        .padding()
    }
}
```

```swift
struct ContentView: View {
    var body: some View {
        VStack {
            Image(systemName: "globe")
                .imageScale(.large)
                .foregroundStyle(.tint)
            Text("Hello, World!")
        }
        .padding()
        .onAppear {
            // Code written here will execute
            //   when your View appears
        }
    }
}
```

View Builder

Logical Code

Printing to Console


Welcome to Xcode
Welcome to Xcode
AppModel
Assets
ContentView
ImmersiveView
Info
ToggleImmersiveSpaceButton
Welcome_to_XcodeApp
Packages
RealityKitContent
Filter
Welcome to Xcode
Apple Vision Pro 4K
Running Welcome to Xcode on Apple Vision Pro 4K
Welcome to Xcode
Welcome to Xcode
ContentView
body
8 import SwiftUI
9 import RealityKit
10 import RealityKitContent
11
12 struct ContentView: View {
13 var body: some View {
14 VStack {
15 Image(systemName: "face.smiling")
16 .imageScale(.large)
17 .foregroundStyle(.tint)
18 Text("Hello!")
19 }
20 .onAppear {
21 print("Hello, World!")
22 }
23 }
24 }
25
26 #Preview(windowStyle: .automatic) {
27 ContentView()
28 .environment(AppModel())
29 }
30
31
ContentView
Hello!
Hello, World!
Auto
Filter
Executable
Canvas
Filter

```
print("Hello, World") // Prints “Hello, World” to the console

let myFavouriteNumber = 42

print(myFavouriteNumber) // Prints “42” to the console
print("My favourite number is \(myFavouriteNumber)”)
// Prints “My favourite number is 42” to the console
```

Constants, Variables & Types

State Variables & Buttons

Binding Variables

Conditional Rendering

When a State value changes, SwiftUI **updates the parts of the view hierarchy that depend on the value**.

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")
        }
        .padding()
    }
}
```

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            Button("Ate Another One") {
                counter += 1
            }
        }
        .padding()
    }
}
```

Buttons

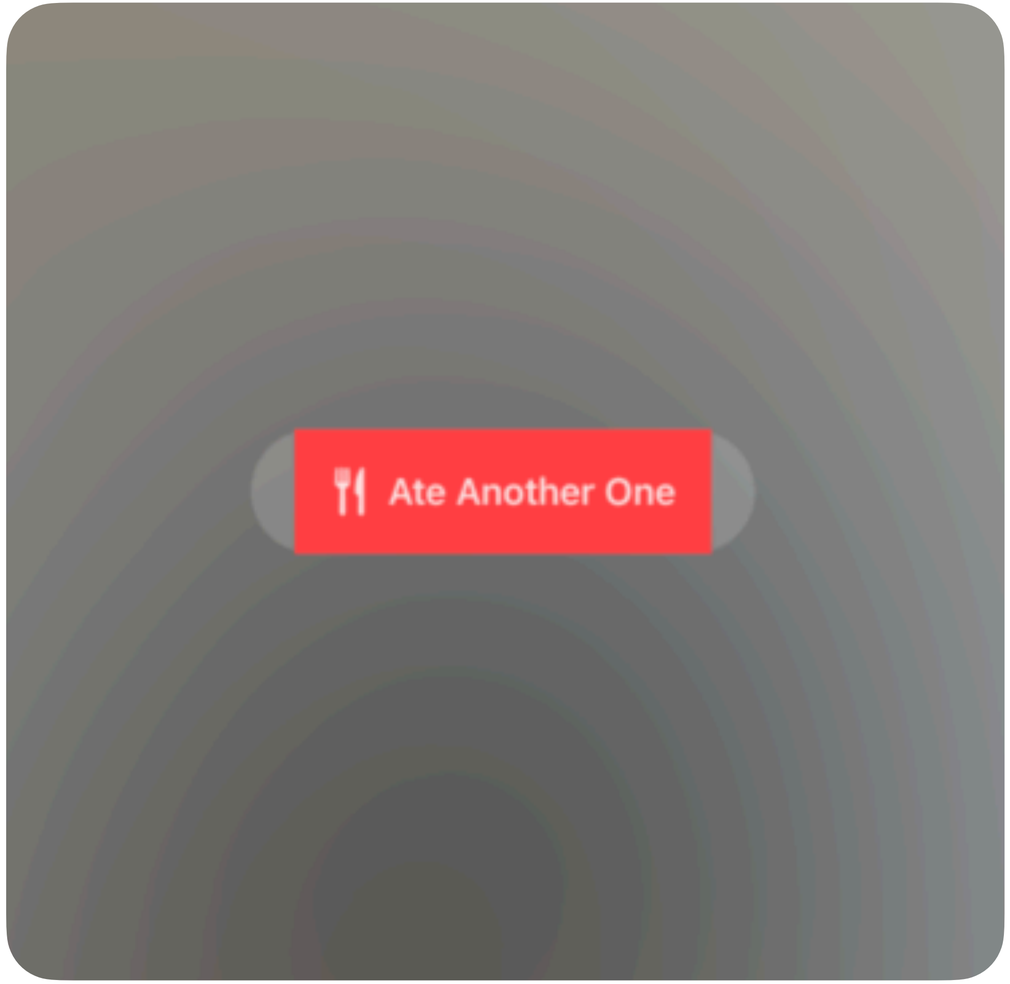
```
Button("Ate Another One") {
    counter += 1
}
```

```
Button {
    counter += 1
} label: {
    Text("Ate Another One")
}
```

```
Button {
    counter += 1
} label: {
    Text("Ate Another One")
        .padding()
}
```

```
Button {
    counter += 1
} label: {
    Text("Ate Another One")
        .padding()
        .background(.red)
}
```

```
Button {
    counter += 1
} label: {
    HStack {
        Image(systemName: "fork.knife")
        Text("Ate Another One")
    }
    .padding()
    .background(.red)
}
```



```swift
Button {
    counter += 1
} label: {
    HStack {
        Image(systemName: "fork.knife")
        Text("Ate Another One")
    }
    .padding()
    .background(.red)
}
```

Logical Code

View Builder

ContentView.swift

Activity

Create a Decrement Button

- Beside the “Ate Another One” button, create another button to decrement the count of cookies
- Try out the `.buttonStyle` modifier and the Button initializer that accepts a View Builder.

15-minute timer has not started.

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            HStack {
                Button("Ate Another One") {
                    counter += 1
                }
                .buttonStyle(.borderedProminent)

                Button {
                    counter -= 1
                } label: {
                    Text("Decrement")
                }
            }
        }
        .padding()
    }
}
```

Constants, Variables & Types

State Variables & Buttons

Binding Variables

Conditional Rendering

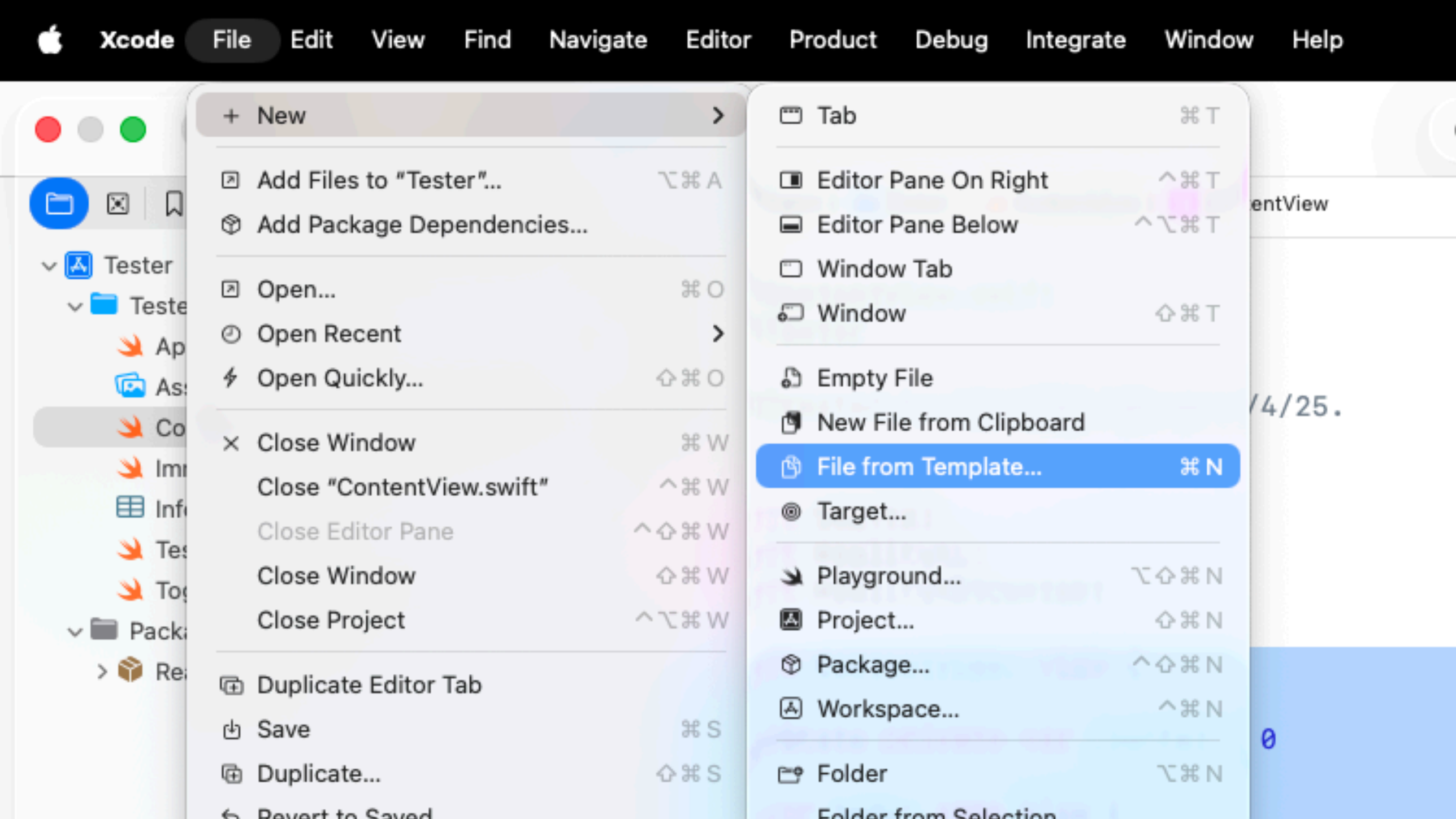
Binding creates a **two-way connection** between a property that stores data, and a view that displays and changes the data.
A binding connects a property to a source of truth stored elsewhere, instead of storing data directly.

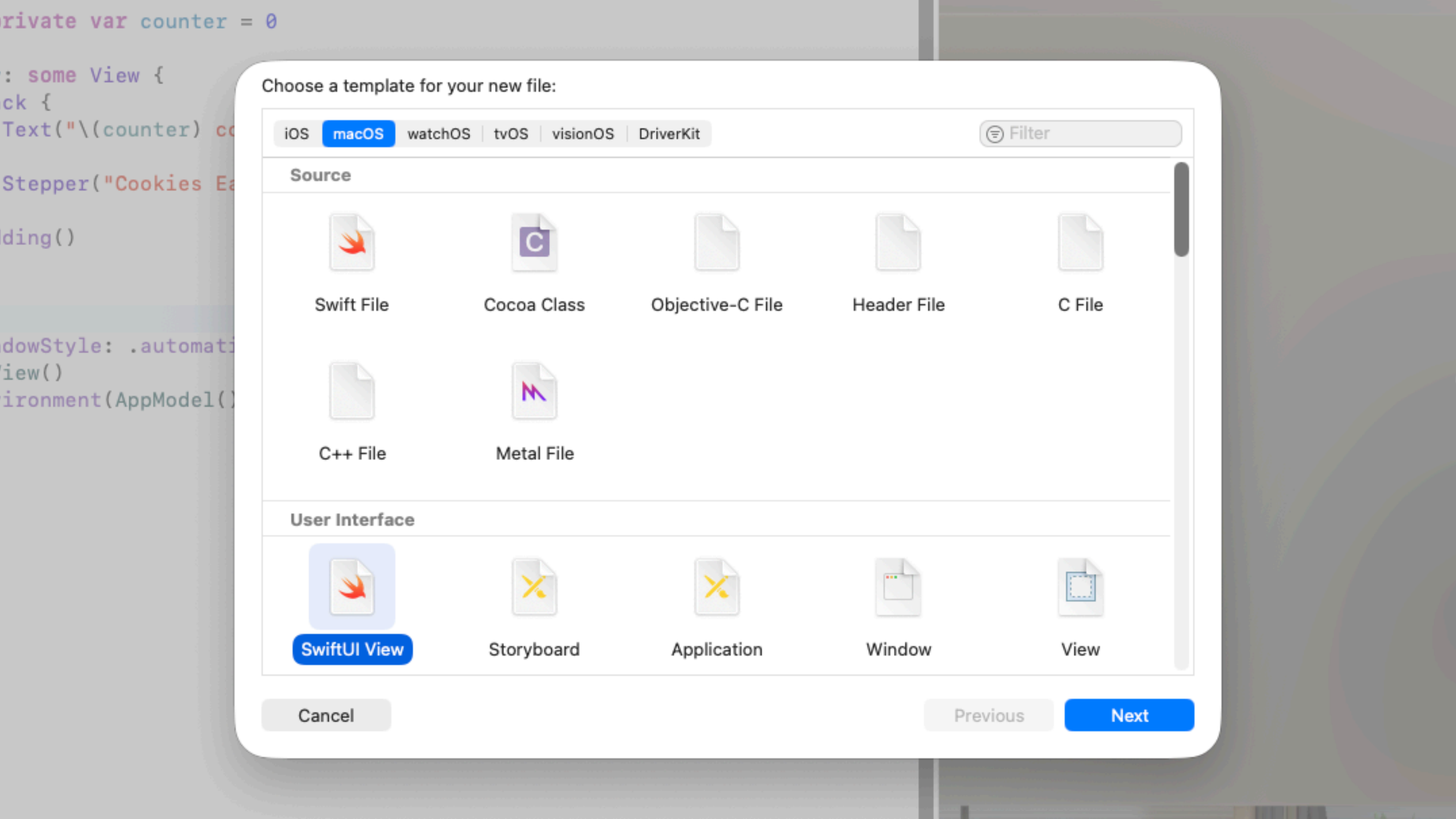
```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            Stepper("Cookies Eaten", value: $counter)
        }
        .padding()
    }
}
```

```
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            Stepper("Cookies Eaten", value: $counter)
        }
        .padding()
    }
}
```

Refactoring Views

Views are designed to be modular, reusable, and **serve a single responsibility**. Small, modular views increases clarity and ease of debugging.


Xcode
File
Edit
View
Find
Navigate
Editor
Product
Debug
Integrate
Window
Help
New
Add Files to "Tester"...
Add Package Dependencies...
Open...
Open Recent
Open Quickly...
Close Window
Close "ContentView.swift"
Close Editor Pane
Close Window
Close Project
Duplicate Editor Tab
Save
Duplicate...
Tab
Editor Pane On Right
Editor Pane Below
Window Tab
Window
Empty File
New File from Clipboard
File from Template...
Target...
Playground...
Project...
Package...
Workspace...
Folder
Tester


rivate var counter = 0
: some View {
ck {
Text("\(counter) co
Stepper("Cookies Ea
lding()
dowStyle: .automati
iew()
ironment(AppModel()
Choose a template for your new file:
iOS
macOS
watchOS
tvOS
visionOS
DriverKit
Filter
Source
Swift File
Cocoa Class
Objective-C File
Header File
C File
C++ File
Metal File
User Interface
SwiftUI View
Storyboard
Application
Window
View
Cancel
Previous
Next

```swift
import SwiftUI

struct CustomStepperView: View {
    var body: some View {
        Text("Hello, World!")
    }
}

#Preview {
    CustomStepperView()
}
```

CustomStepperView.swift

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            Stepper("Cookies Eaten", value: $counter)
        }
        .padding()
    }
}
```

ContentView.swift

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            CustomStepperView()

            Stepper("Cookies Eaten", value: $counter)
        }
        .padding()
    }
}
```

ContentView.swift


ContentView
0 cookies eaten!
Hello, World.
CustomStepperView
Cookies Eaten
−
+

Declaring Binding Variables

```swift
struct CustomStepperView: View {
    var body: some View {
        Text("Hello, World!")
    }
}

#Preview {
    CustomStepperView()
}
```

CustomStepperView.swift

```swift
struct CustomStepperView: View {

    @Binding var counter: Int

    var body: some View {
        Text("Hello, World!")
    }
}

#Preview {
    CustomStepperView()
}
```

CustomStepperView.swift

```swift
struct CustomStepperView: View {

    @Binding var counter: Int

    var body: some View {
        HStack {
            Button("Increment") {
                counter += 1
            }

            Button("Decrement") {
                counter -= 1
            }

            Button("Reset") {
                counter -= 1
            }
        }
    }
}
```

CustomStepperView.swift

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            CustomStepperView()

            Stepper("Cookies Eaten", value: $counter)
        }
        .padding()
    }
}
```

ContentView.swift

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            CustomStepperView(counter: $counter)
        }
        .padding()
    }
}
```

ContentView.swift

Constants, Variables & Types

State Variables & Buttons

Binding Variables

Conditional Rendering

Conditional Rendering allows you to **show and hide a view**, with a Boolean condition.

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            if counter >= 10 {
                Text("That's a lot of cookies!")
            }

            CustomStepperView(counter: $counter)
        }
        .padding()
    }
}
```

ContentView.swift

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            if counter >= 10 {
                Text("That's a lot of cookies!")
            } else if counter >= 5 {
                Text("That's a healthy amount of cookies")
            }

            CustomStepperView(counter: $counter)
        }
        .padding()
    }
}
```

ContentView.swift

```swift
struct ContentView: View {

    @State private var counter = 0

    var body: some View {
        VStack {
            Text("\(counter) cookies eaten!")

            if counter >= 10 {
                Text("That's a lot of cookies!")
            } else if counter >= 5 {
                Text("That's a healthy amount of cookies")
            } else {
                Text("Eat more!")
            }

            CustomStepperView(counter: $counter)
        }
        .padding()
    }
}
```

ContentView.swift