Building Apps with SwiftUI

CMSC 436

Decluttering the UI with Tabs

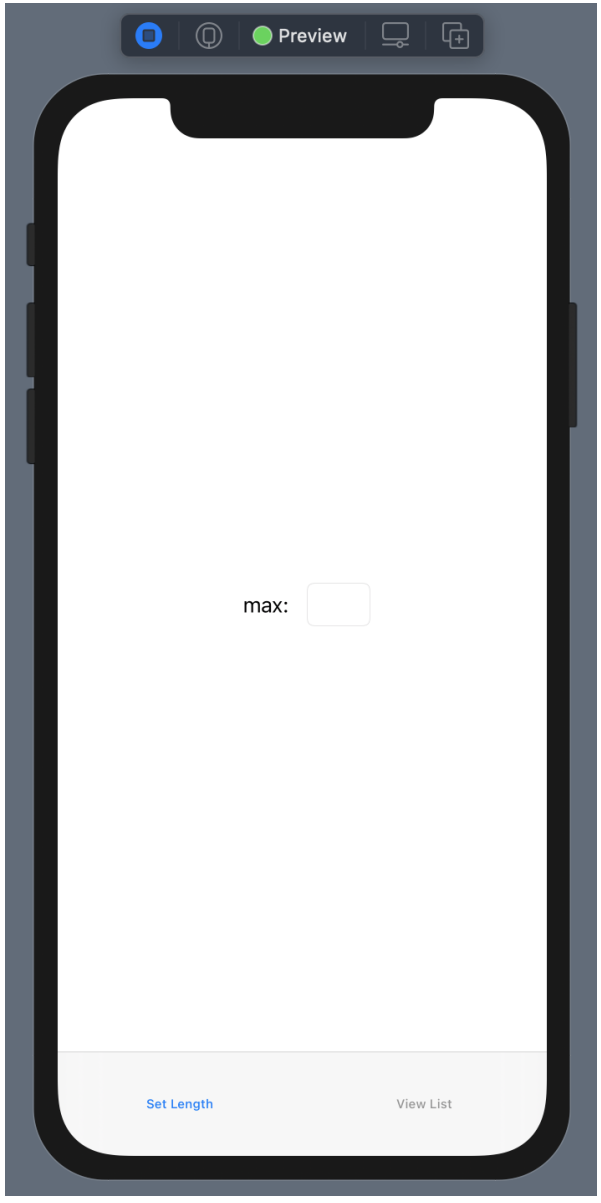
```
var body: some View {
    TabView {
        HStack {
            Text("max:").frame(width:50,height:20)//.padding()
            TextField("",text:$mvText) {
                _ in if let i = Int(mvText) { vals.setMax(i) }
            }
            .frame(width: 50, height: 20, alignment: .center)
            .textFieldStyle(RoundedBorderTextFieldStyle())
        }.tabItem { Text("Set Length") }
        MyScroll(vals.maxVal).padding().tabItem { Text("View List") }
    }
}
```

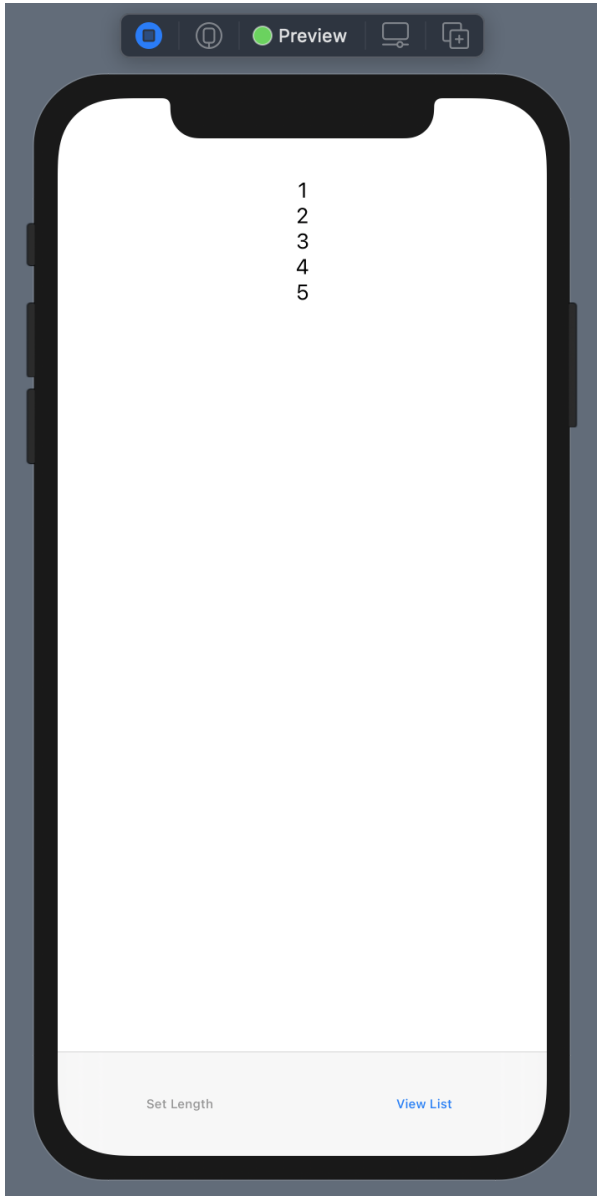
Embed tabs in a `TabView`

`tabItem()` method lets us provide a label

p

Tabbed Version of the UI

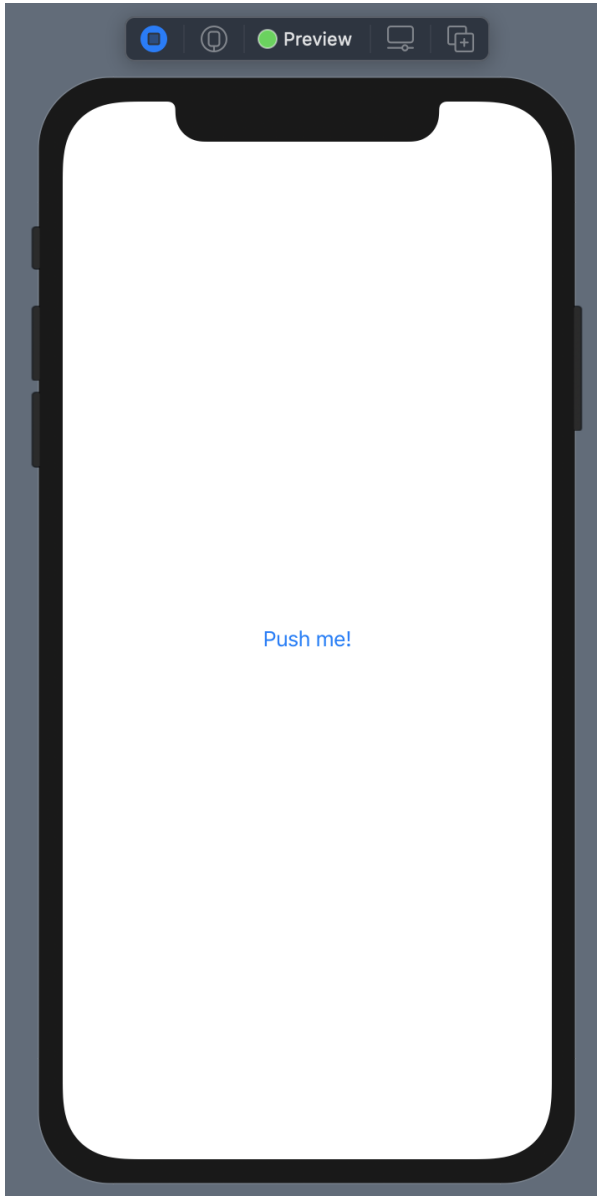


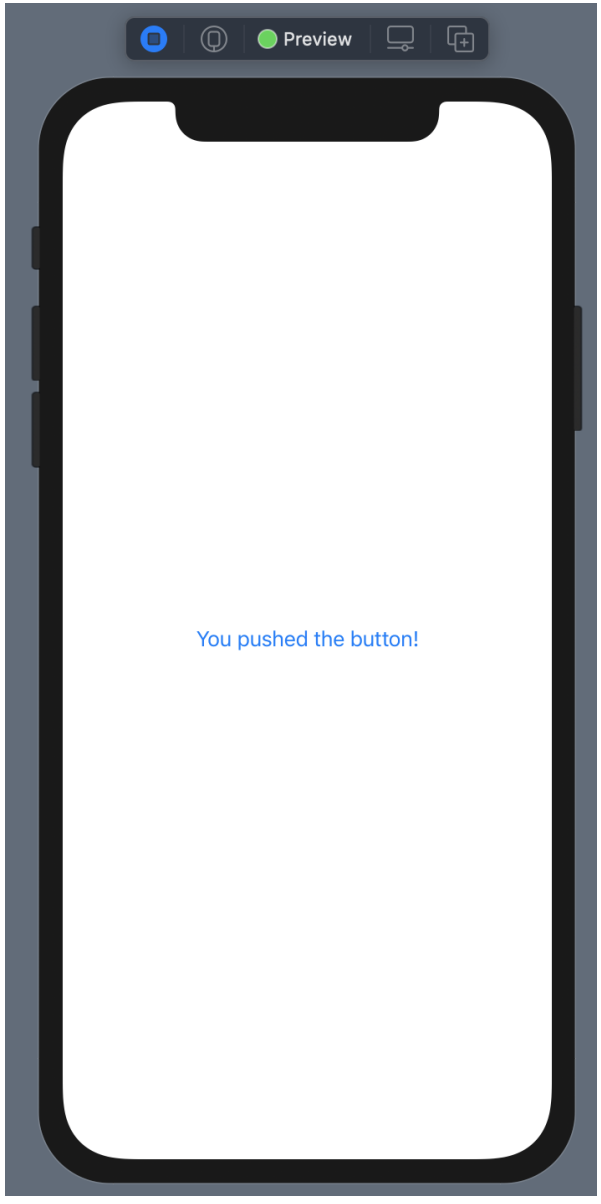


Buttons

```
struct ButtonView: View {
   @State private var buttonText = "Push me!"

   var body: some View {
      Button(action: f) {
         Text(buttonText)
      }
   }

   func f() {
      buttonText = "You pushed the button!"
   }
}
```

Testing ButtonView





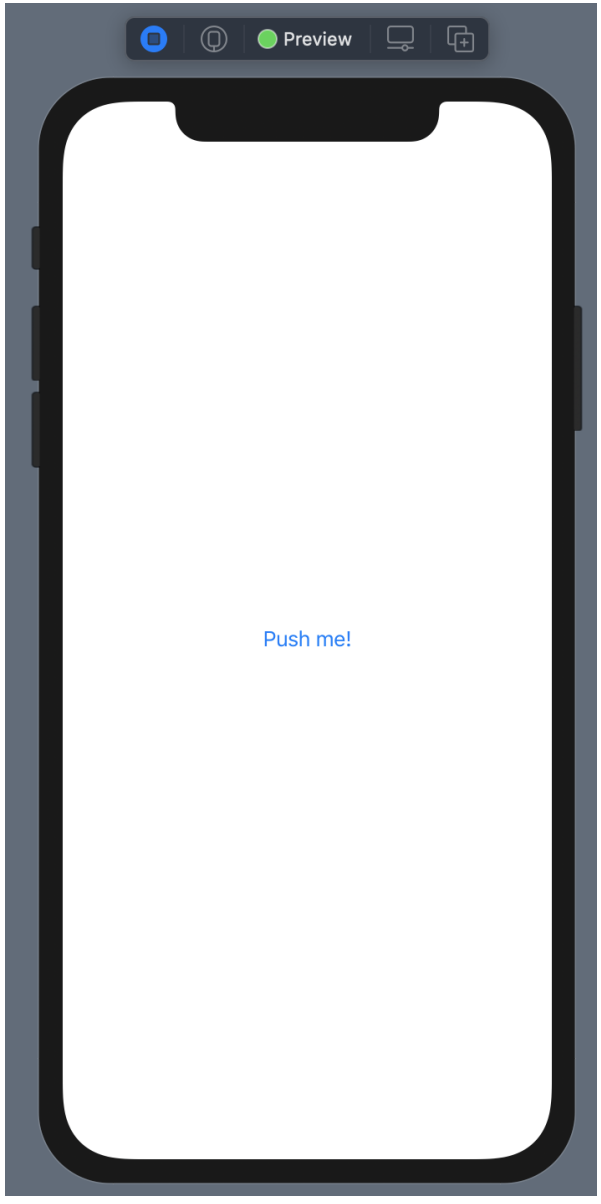
Buttons and Alerts

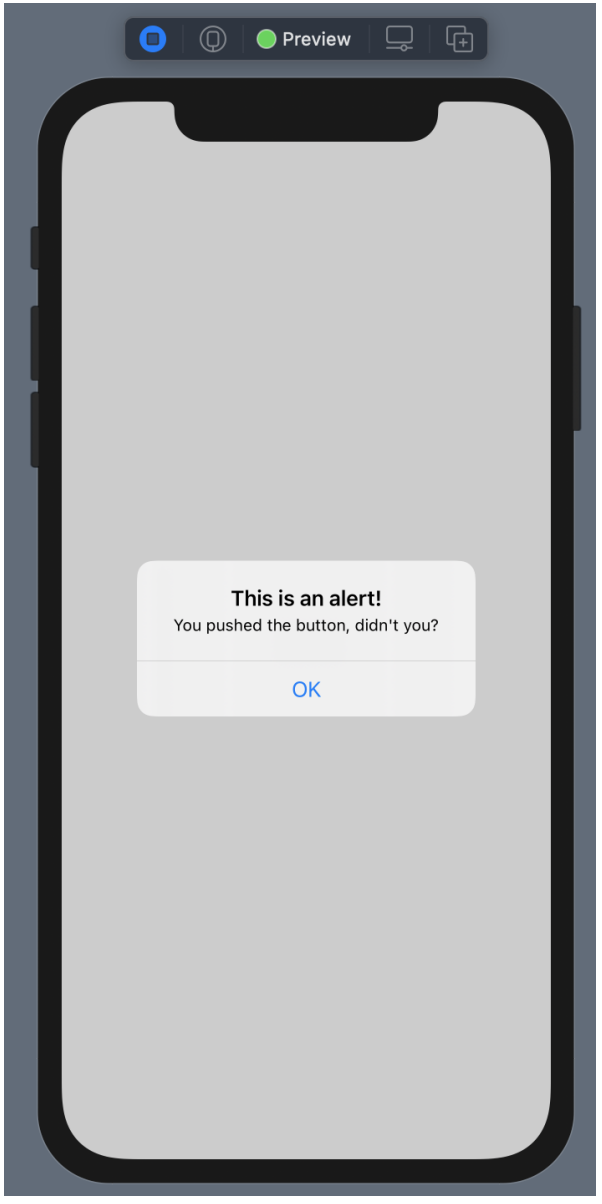
```
struct AlertingButtonView: View {
   @State private var showAlert = false

   var body: some View {
      Button("Push me!") {
         showAlert = true
      }.alert(isPresented: $showAlert) {
         Alert(
            title: Text("This is an alert!"),
            message: Text("You pushed the button, didn't you?")
         )
      }
   }
}
```

All `Views` have an `alert()` method

`Alert` based on app state (errors, scheduled annoucements, etc.)

Testing `AlertingButtonView`

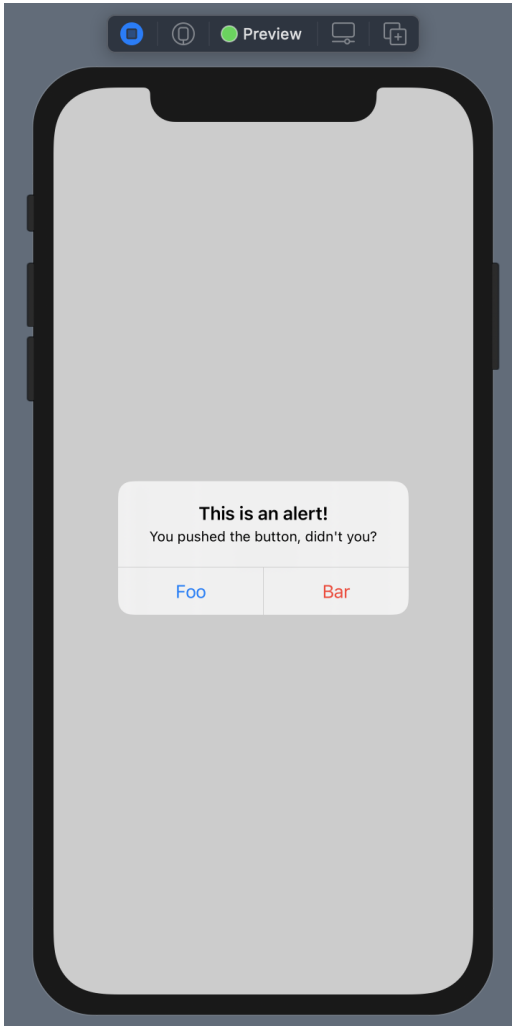


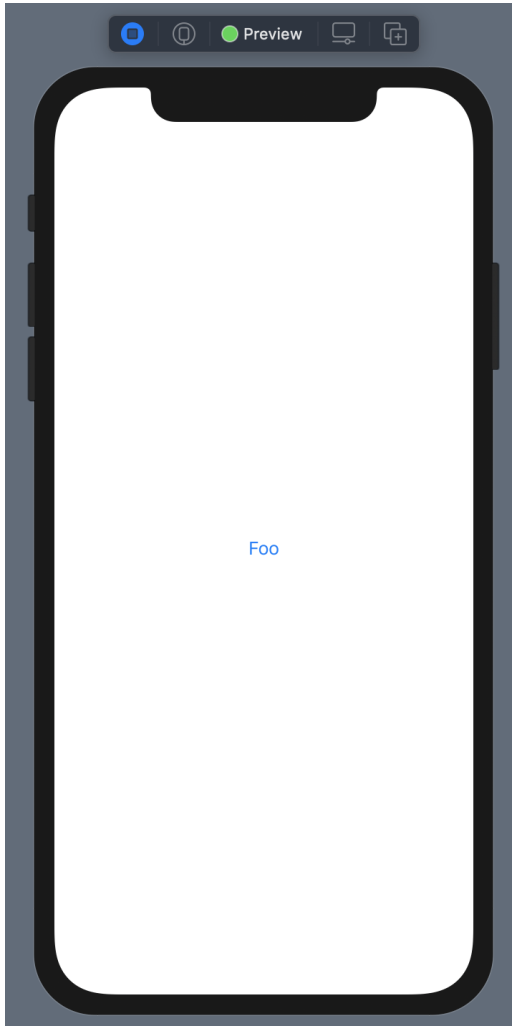


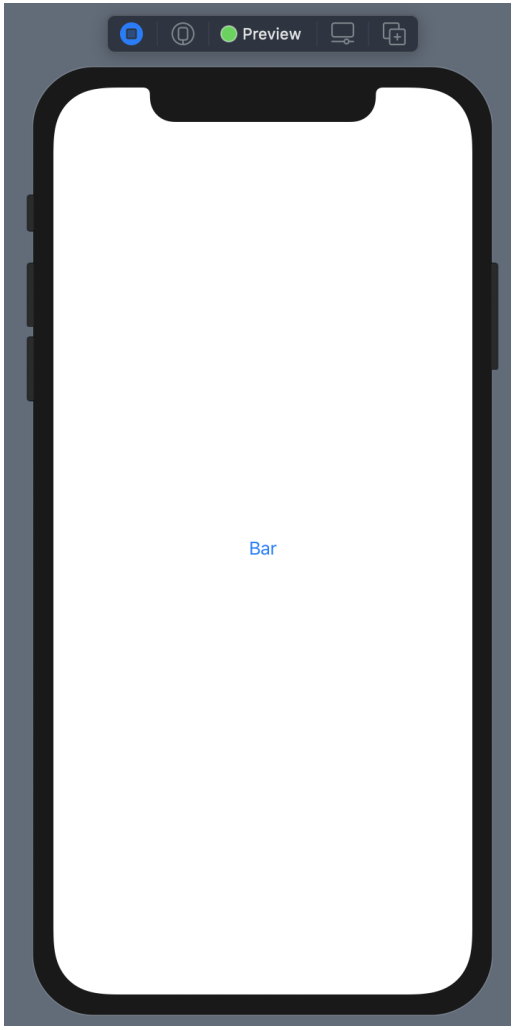
Buttons and Alerts (and Alert Buttons)

```
struct CustomAlertingButtonView: View {
   @State private var showAlert = false
   @State private var buttonText = "Push me!"

   var body: some View {
       Button(buttonText) {
          showAlert = true
       }.alert(isPresented: $showAlert) {
          Alert(
             title: Text("This is an alert!"),
             message: Text("You pushed the button, didn't you?"),
             primaryButton: .default( Text("Foo"),
                                      action: {
                                         buttonText = "Foo"
                                      } ),
             secondaryButton: .destructive( Text("Bar"),
                                            action: {
                                               buttonText = "Bar"
                                            } )
          )
       }
   }
}
```

Testing `CustomAlertingButtonView`







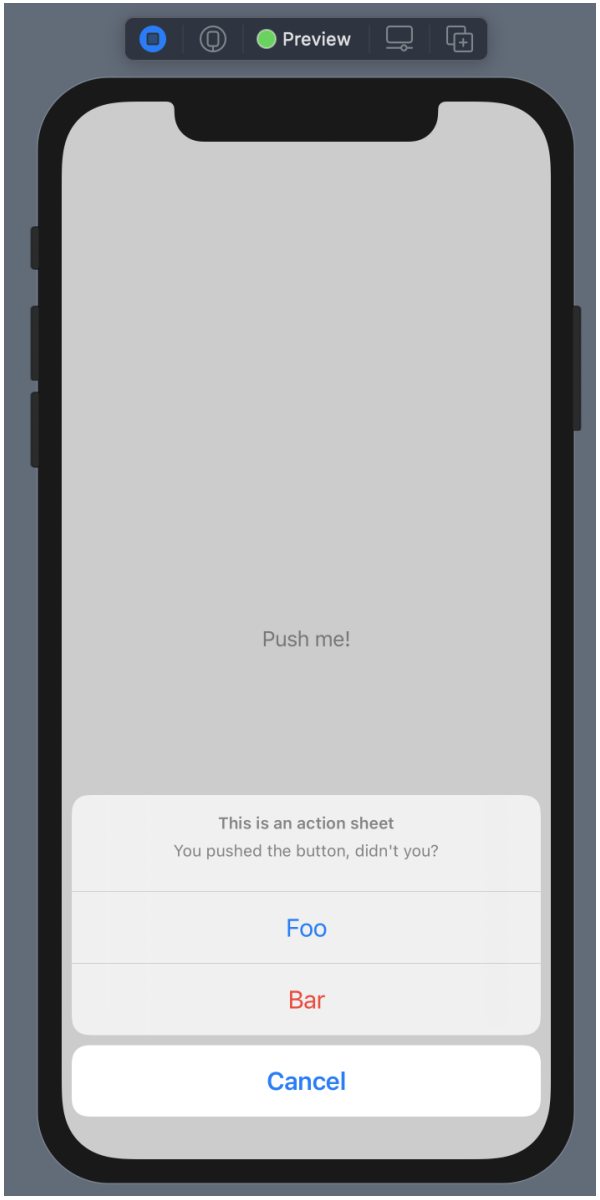
Action Sheets

Modal (like Alerts)

Based on user actions

```
struct ActionSheetView: View {
   @State private var showActionSheet = false
   @State private var buttonText = "Push me!"

   var body: some View {
      Button(buttonText) {
         showActionSheet = true
      }.actionSheet(isPresented: $showActionSheet) {
         ActionSheet(
            title: Text("This is an action sheet"),
            message: Text("You pushed the button, didn't you?"),
            buttons: [
               .cancel(),
               .default(Text("Foo"),
                        action: { buttonText = "Foo" } ),
               .destructive(Text("Bar"),
                            action: { buttonText = "Bar" } )
            ]
         )
      }
   }
}
```

Testing `ActionSheetView`



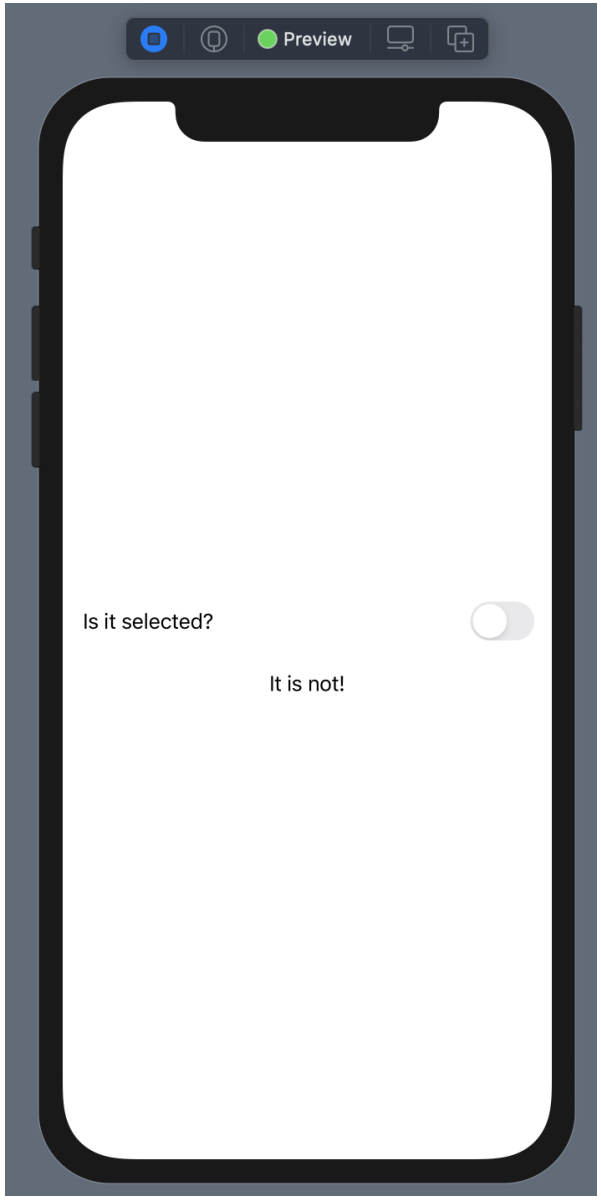
- Appears at bottom
- System might re-order the buttons
- As many buttons as you need
- Button actions can be methods, functions, or closures (like with `Alerts` and `Buttons`)

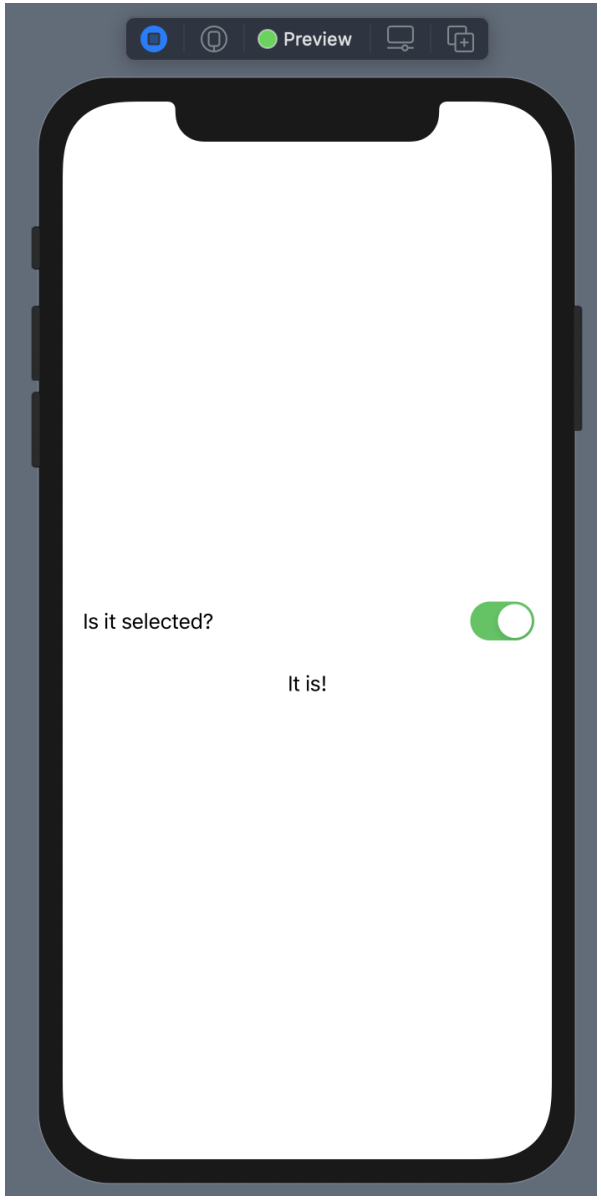
Toggles

Sometimes we need a boolean

```
struct ToggleView: View {
    @State private var selected = false
    private var yes = "It is!"
    private var no = "It is not!"

    var body: some View {
        VStack{
            Toggle(isOn: $selected) {
                Text("Is it selected?")
            }.padding()
            Text("\(selected ? yes : no )")
        }
    }
}
```

Testing ToggleView



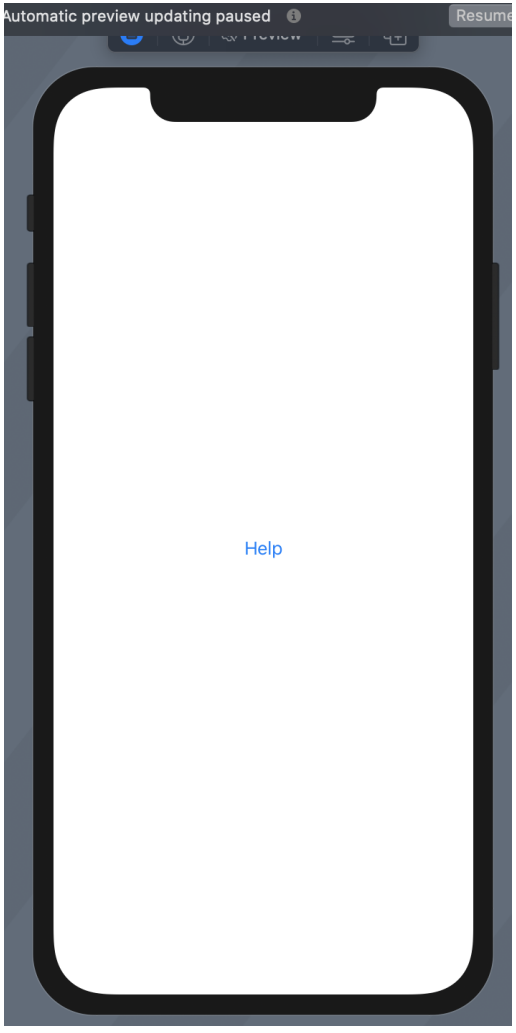


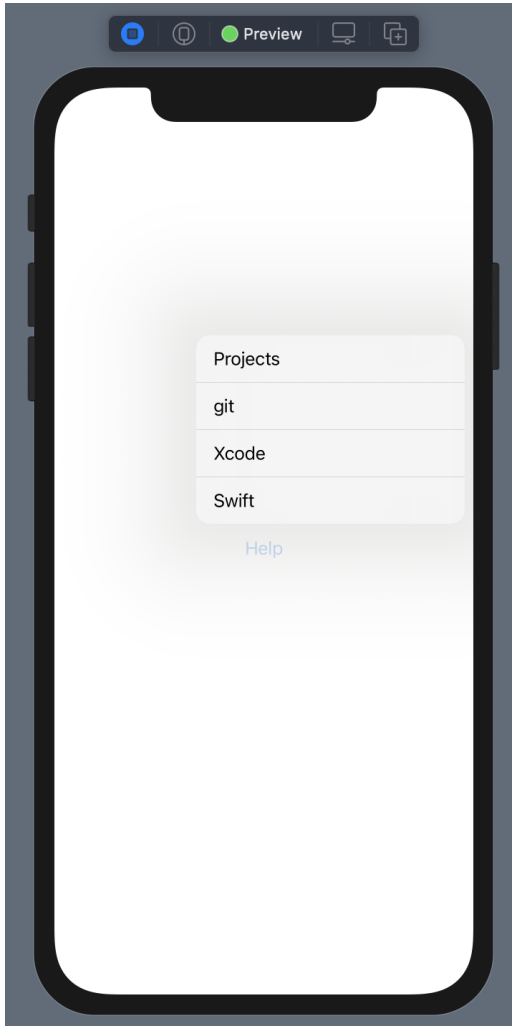
Menus

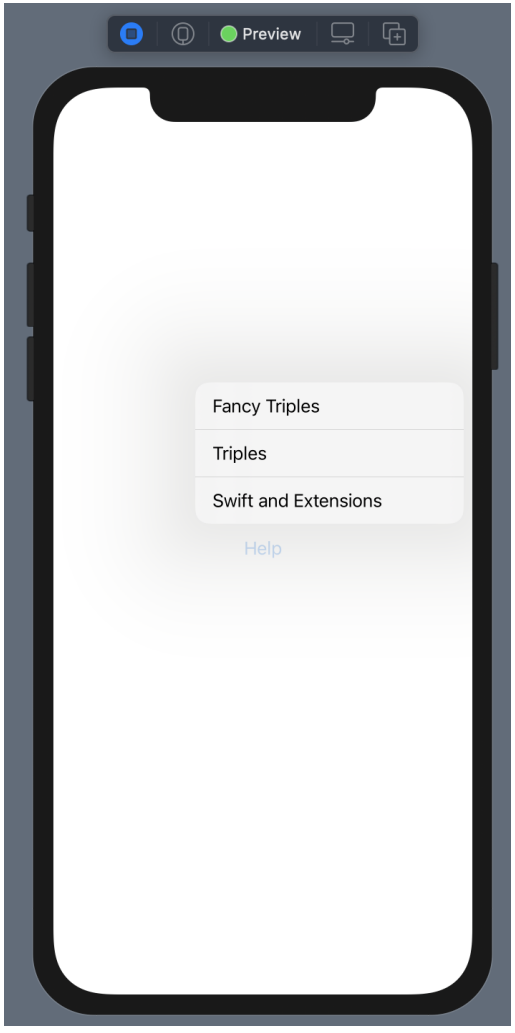
```
struct MenuView: View {
    var body: some View {
        Menu {
            Button("Swift", action: {})
            Button("Xcode", action: {})
            Button("git", action: {})
            Menu("Projects") {
                Button("Swift and Extensions", action: {})
                Button("Triples", action: {})
                Button("Fancy Triples", action: {})
            }
        } label: {
            Text("Help")
        }
    }
}
```

This shows two of the ways of creating a Menu

Testing `MenuView`







Note that the order of the options is reversed!

Gestures

The stuff you're used to:

- **Tap** (`TapGesture`)
- **Long press** (`LongPressGesture`)
- **Drag** (`DragGesture`)
- Magnify (`MagnificationGesture`)
- Rotate (`RotationGesture`)

All of these have:

- `onEnded()`
- `updating()`

Most of them (excluding Tap) have:

- `onChanged()`

Tap Gestures

Anything can be a button!

`init()` takes an optional count
  $\Rightarrow$ You have to tap this many times to activate it

`onEnded()` takes a closure `() -> Void`
  $\Rightarrow$ This is executed when activated