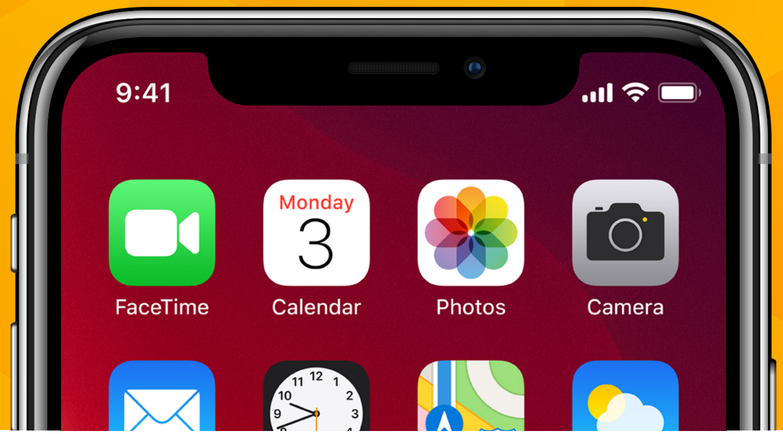



# iOS Application Development

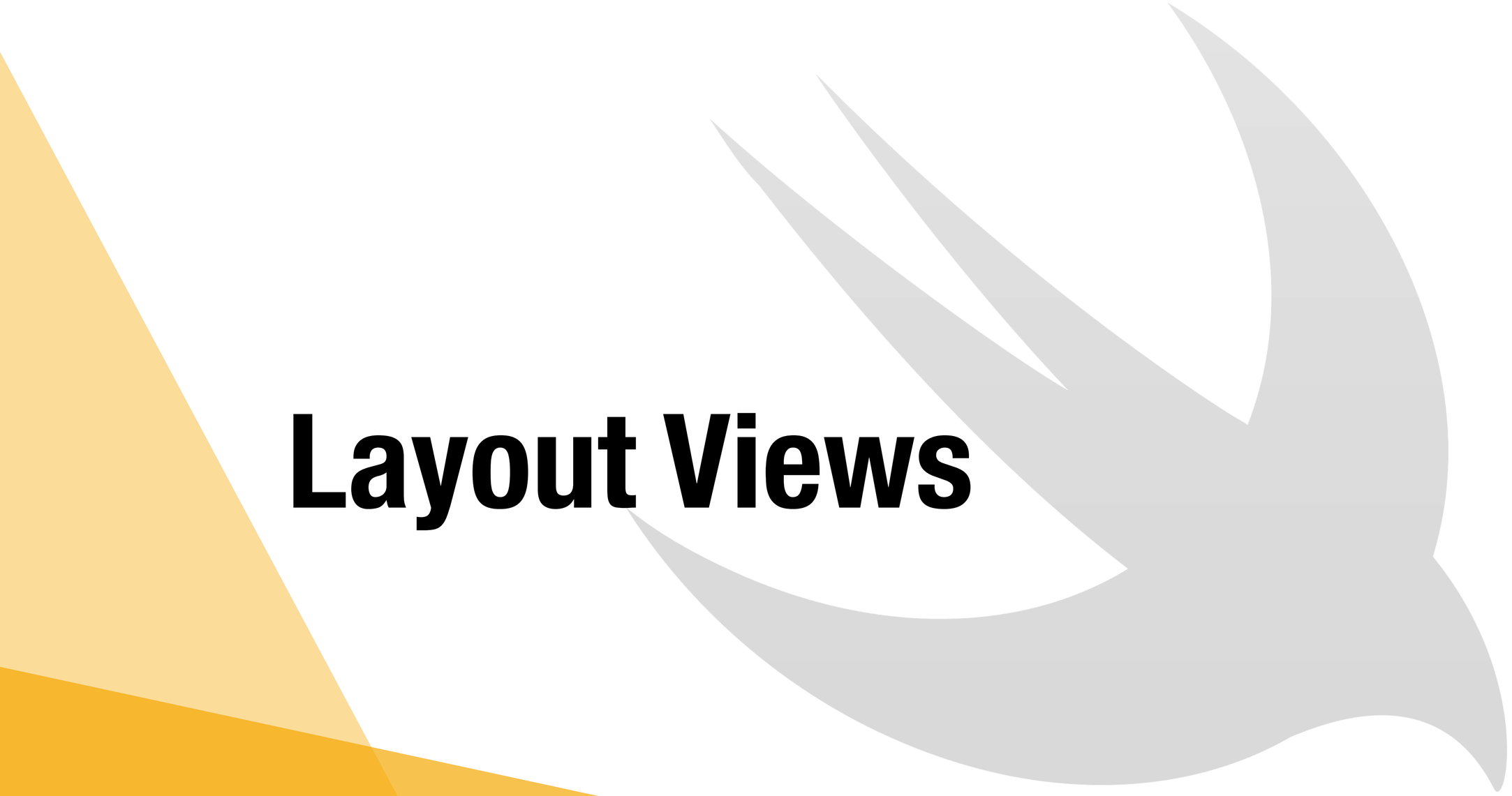
## Lecture 11: SwiftUI Continued

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

WS ’22/’23 • hci.rwth-aachen.de/ios

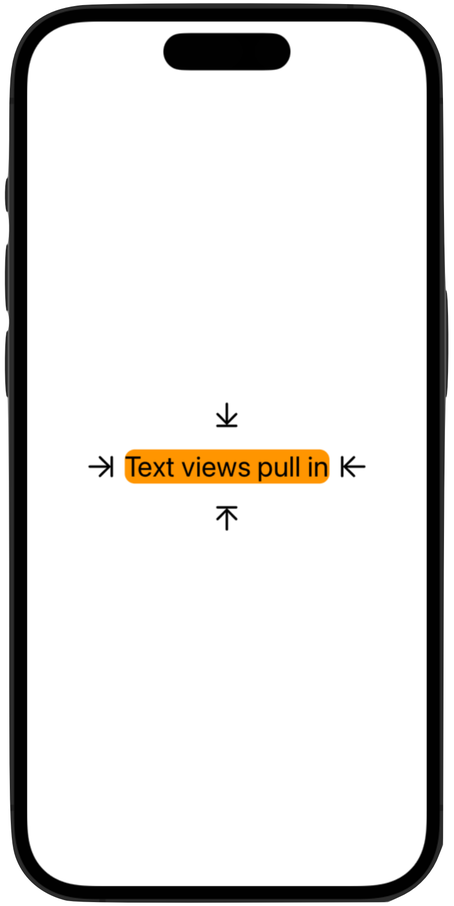
RWTH AACHEN UNIVERSITY

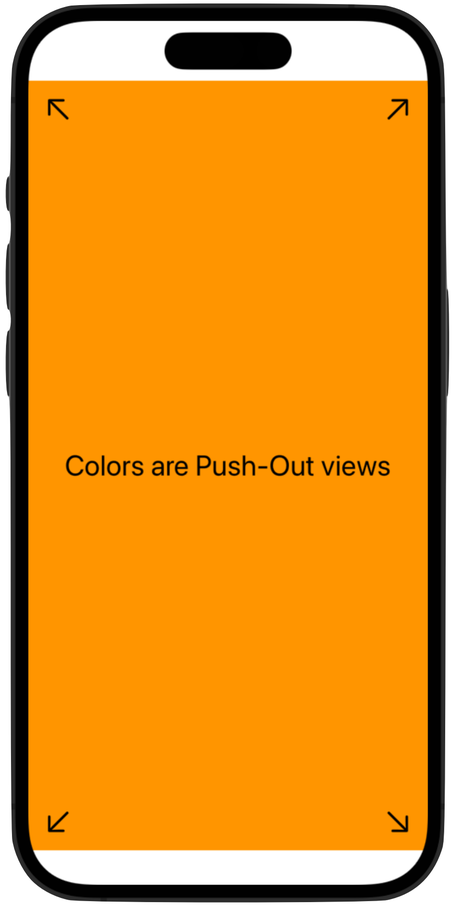
# Layout Views



# Layout Behavior

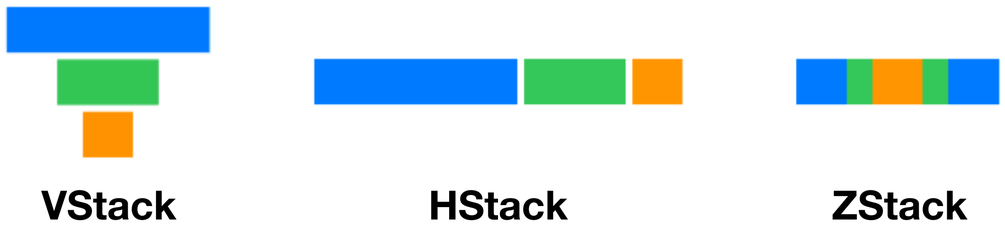
- Two behaviors of views



# VStack, HStack & ZStack

- Pull-in container views
- Like all layout views, can only contain up to 10 items
  - Group{} to create larger lists
- LazyVStack & LazyHStack load items only if visible
  - Push-out containers

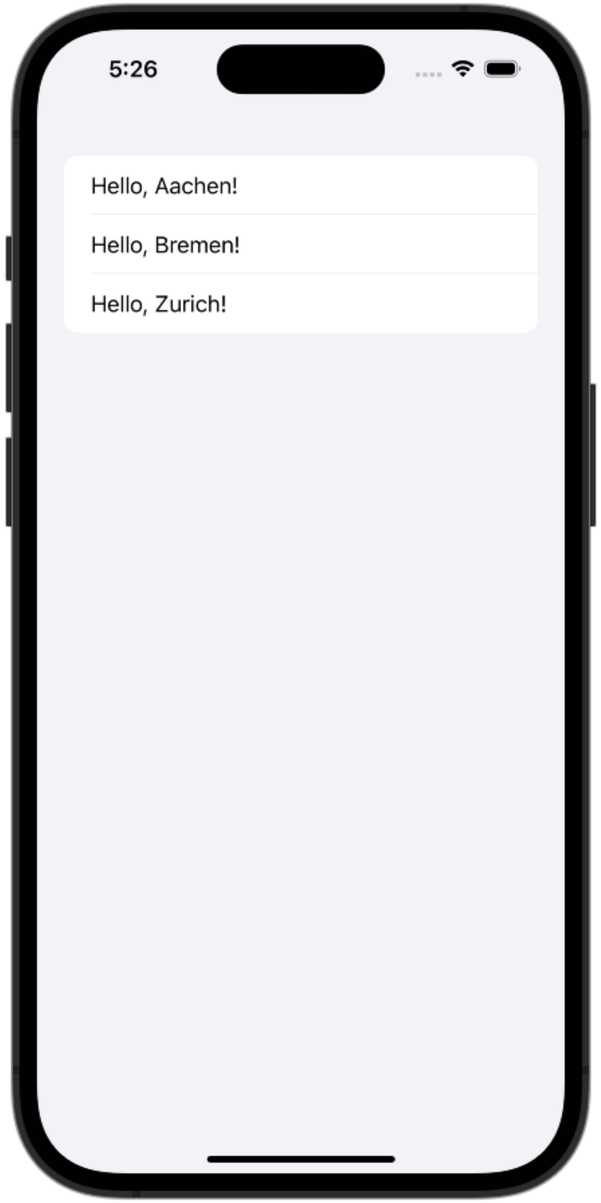



```
//  Replace VStack with HStack or HStack here
VStack(alignment: .center, spacing: 5) {
    Text(" ")
        .frame(width: 150)
        .background(.blue)
    Text(" ")
        .frame(width: 75)
        .background(.green)
    Text(" ")
        .frame(width: 37)
        .background(.orange)
}
```

# List

- Vertically scrolling list
- Very efficient
- Items can be different View types
- Multiple selection, Delete via swipe optional
- `Group{}` helps with more than 10 items
- `Section{}` creates a separated group with an optional title

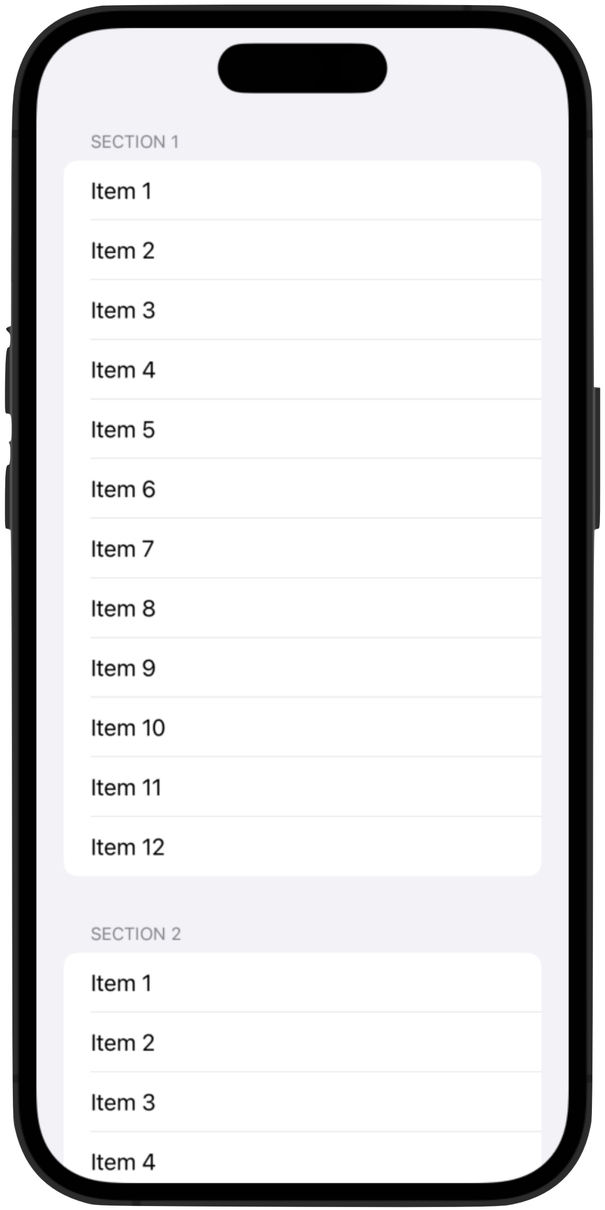
```
struct ContentView: View {
    var body: some View {
        List {
            Text("Hello, Aachen!")
            Text("Hello, Bremen!")
            Text("Hello, Zurich!")
        }
    }
}
```



# Form

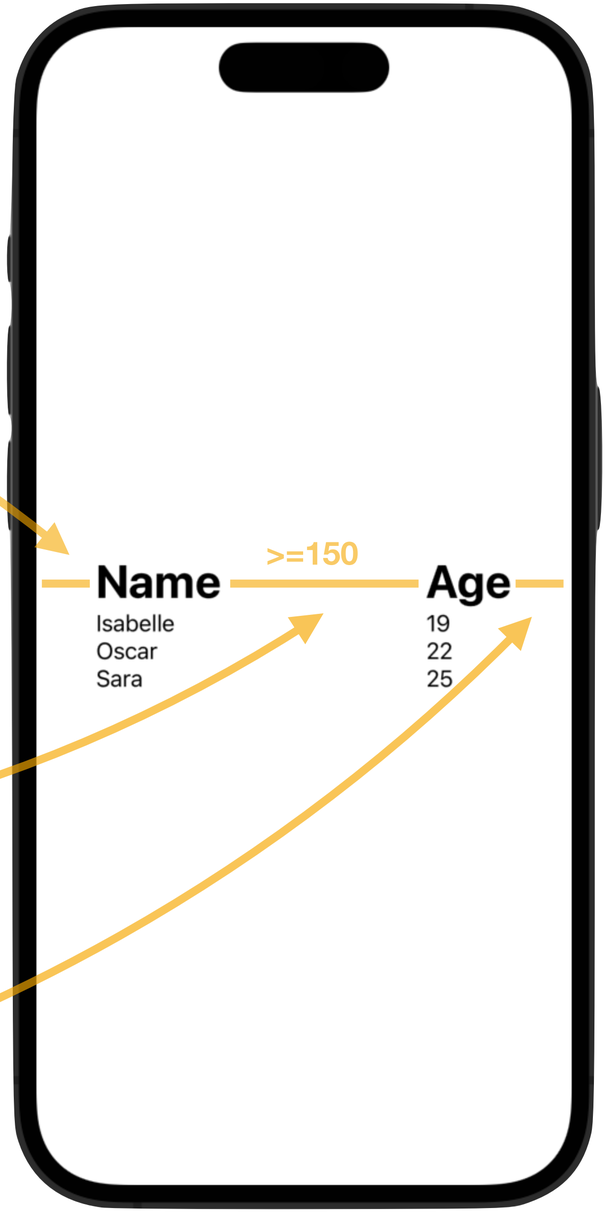
- Similar to List
- Both use UITableView in iOS
- Use Form to group controls used for data entry, like in settings or inspectors
- Used, e.g., in the iOS Settings app

```
struct ContentView: View {
    var body: some View {
        Form {
            Section("Section 1"){
                Group{
                    Text("Item 1")
                    Text("Item 2")
                    Text("Item 3")
                    Text("Item 4")
                    Text("Item 5")
                    Text("Item 6")
                    Text("Item 7")
                    Text("Item 8")
                    Text("Item 9")
                    Text("Item 10")
                }
                Group{
                    Text("Item 11")
                    Text("Item 12")
                }
            }
            Section("Section 2"){
                Text("Item 1")
                Text("Item 2")
                Text("Item 3")
                Text("Item 4")
            }
        }
    }
}
```



# Spacer

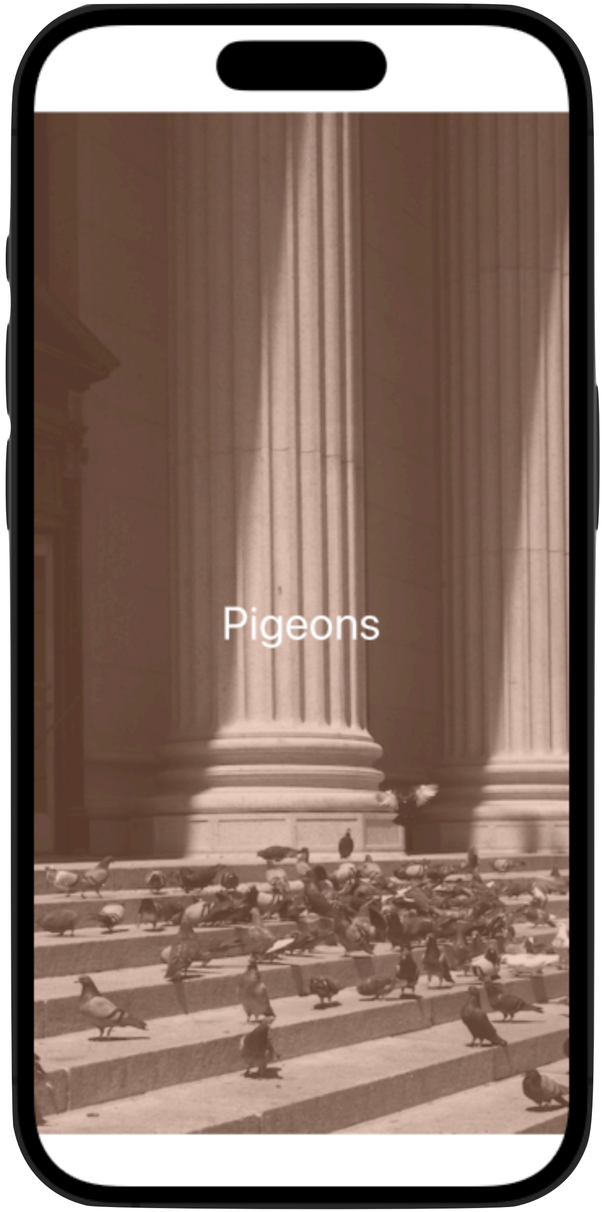
```
struct ContentView: View {
    var body: some View {
        HStack {
            Spacer()
            VStack(alignment: .leading) {
                Text("Name")
                    .font(.largeTitle)
                    .bold()
                Text("Isabelle")
                Text("Oscar")
                Text("Sara")
            }
            Spacer(minLength: 150)
            VStack(alignment: .leading) {
                Text("Age")
                    .font(.largeTitle)
                    .bold()
                Text("19")
                Text("22")
                Text("25")
            }
            Spacer()
        }
    }
}
```



# Layers

- Add layers on top and behind views

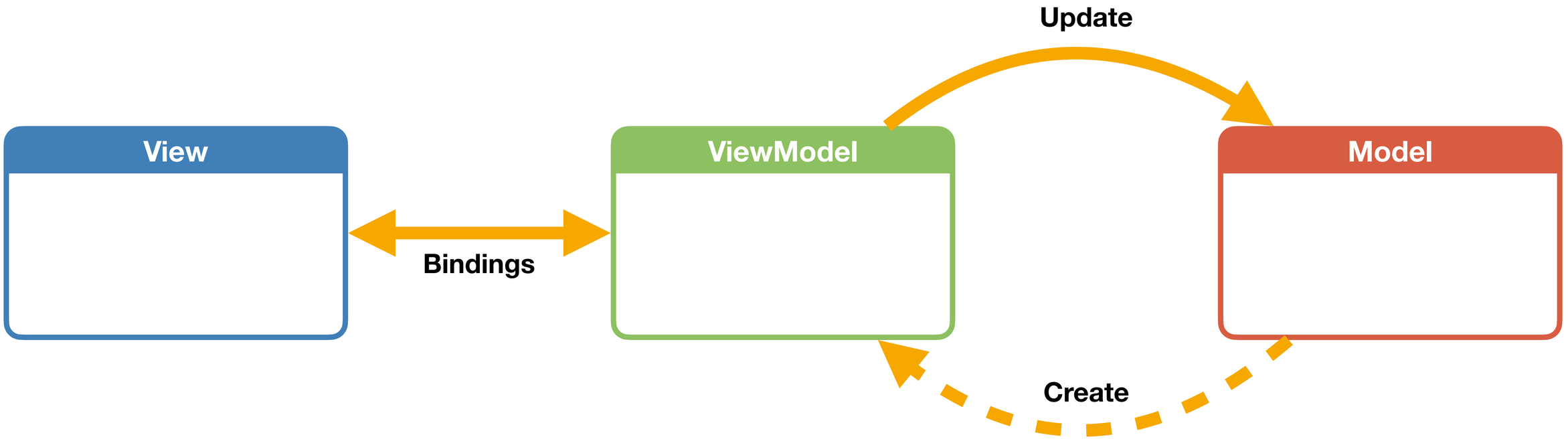
```
struct ContentView: View {
    var body: some View {
        Image("pigeons")
            .frame(maxWidth: .infinity)
            .opacity(0.6) // Make image only 60% solid
            .background(Color.red.opacity(0.4)) // Layer behind image
            .background(Color.yellow.opacity(0.4)) // Layer behind red
            .background(Color.blue.opacity(0.4)) // Layer behind yellow
            .overlay(
                Text("Pigeons")
                    .foregroundColor(.white)
                    .font(.largeTitle)
            ) // Layer on top of image
            .clipped()
    }
}
```



# MVVM: Model / View / ViewModel

- MVC Pros: Separation of concerns, wide adoption, clear data flow
- MVC Cons: "MVC = Massive ViewController", ViewControllers hard to test (UI)
- Solution: Presentation Model [Fowler 2004]: Store app state (ViewModel) independently of UI (View)
- But requires tons of glue code between ViewModel and View
- Solution: **Bindings** (like Cocoa Bindings in Mac OS X since 2003, but not iOS)
- Adopted by Microsoft's Windows Presentation Foundation in 2005 as "MVVM"
- Natural choice for SwiftUI, testability, smaller Views

# MVVM Paradigm

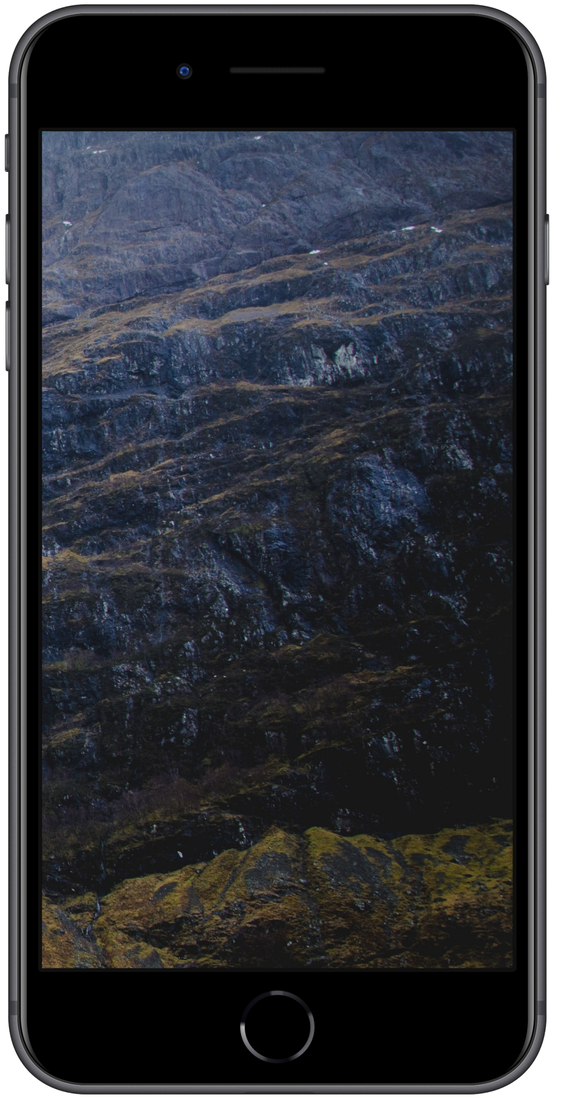



"***MVVM*** *is the modern improvement of MVC for declarative programming*"


Avocado Toast
Bread: Wheat
Spread: None
Avocado: Sliced
Mashed
Add Salt
Add Red Pepper
Quantity: 1
Order
MacBook Pro
9:41
Avocado Toast
Bread Wheat
Spread None
Avocado Sliced
Include Salt
Include Red Pepper
Quantity: 1
Order
Avocado Toast 9:41
Salt
Red Pepper
Order

# Apps on iOS and macOS

| View | View |
| --- | --- |
| AppKit | UIKit |
| **Controller** | **Controller** |
| AppKit | UIKit |
| **Model** | |
| Foundation, CoreData | |



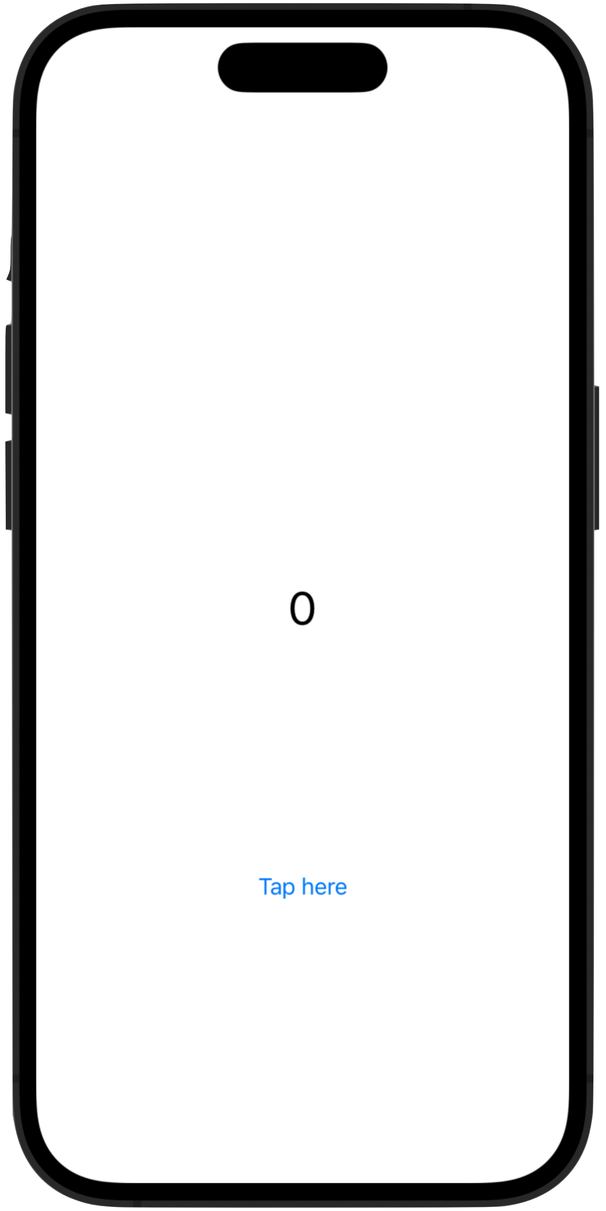
# SwiftUI View Names

| UIKit | SwiftUI |
|---|---|
| UITableView | List |
| UICollectionView | No direct equivalent (alternatives: LazyVStack, LazyHStack,…) |
| UILabel | Text |
| UITextField | TextField |
| UITextField with isSecureTextEntry set to true | SecureField |
| UITextView | TextEditor |
| UISwitch | Toggle |
| UISlider | Slider |
| UIButton | Button |
| UINavigationController | NavigationView (softly deprecated → NavigationStack) |
| UIAlertController with style .alert | Alert |
| UIAlertController with style .actionSheet | ActionSheet |
| UIStackView with horizontal axis | HStack |
| UIStackView with vertical axis | VStack |
| UIImageView | Image |
| UISegmentedControl | SegmentedControl |
| UIStepper | Stepper |
| UIDatePicker | DatePicker |
| NSAttributedString | Incompatible with SwiftUI; use Text instead. |

# Variables & Loops in SwiftUI

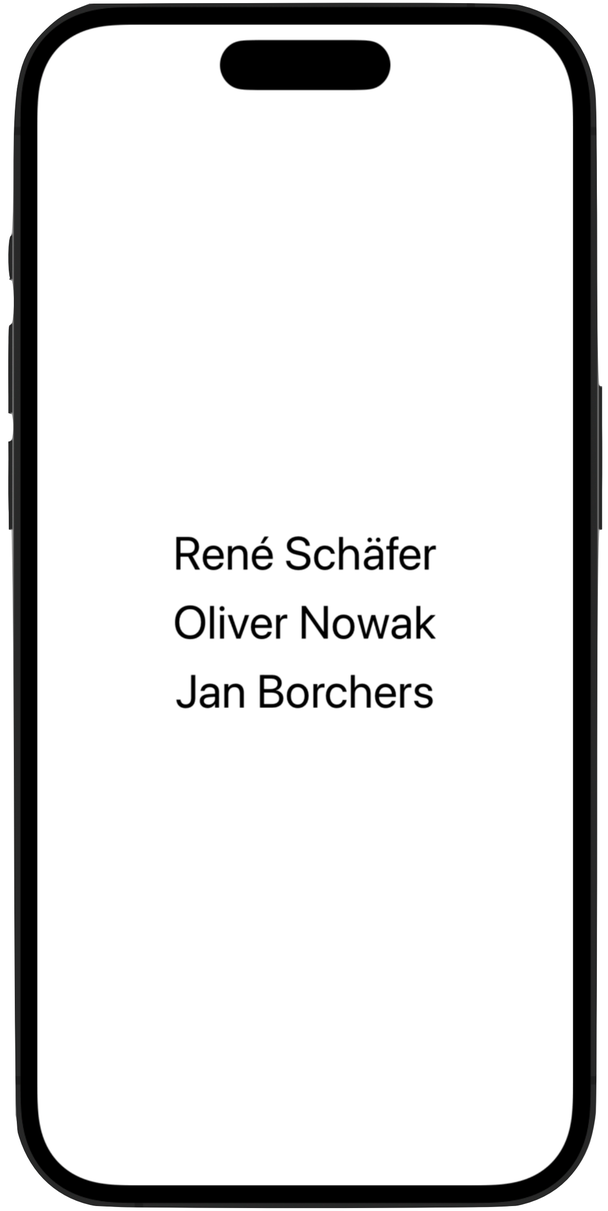
# Variables in SwiftUI

```
struct ContentView: View {
    var tapCount = 0

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\(tapCount)")
                .font(.largeTitle)
            Spacer()
            Button("Tap here") {

            }
            Spacer()
        }

    }
}
```



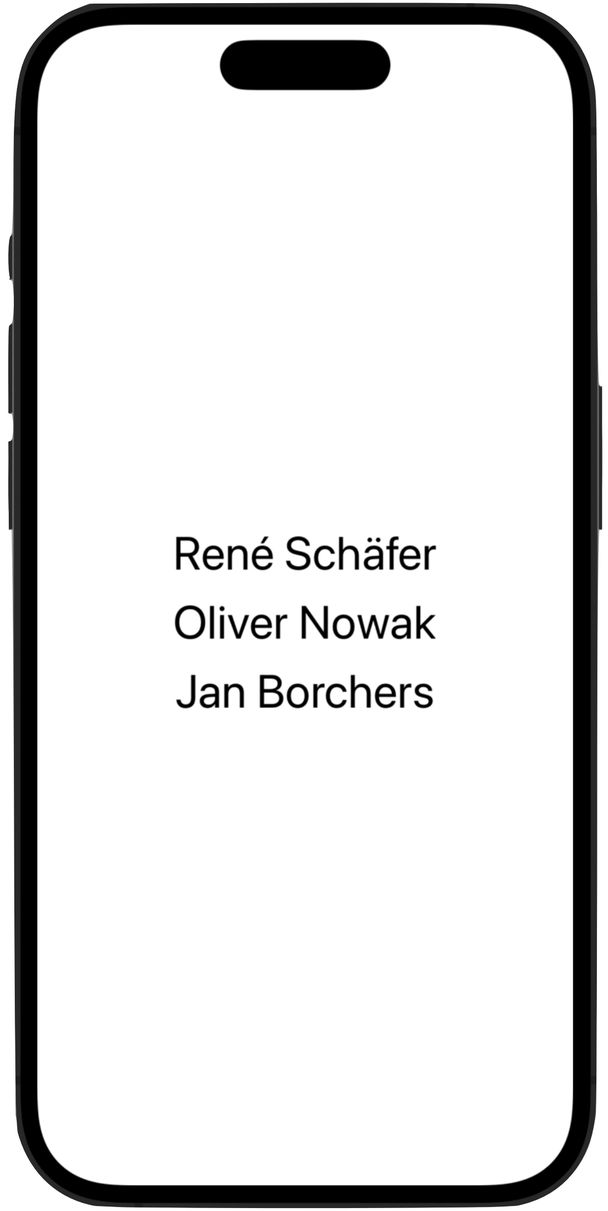
# Variables in SwiftUI

```
struct ContentView: View {
    let peopleInIOS = ["René Schäfer", "Oliver Nowak",
                       "Jan Borchers"]

    var body: some View {
        LazyVStack(spacing: 10) {
            ForEach(0 ..< 3){ index in
                Text(peopleInIOS[index])
                    .font(.largeTitle)
            }
        }
    }
}
```



# Variables in SwiftUI

```
struct ContentView: View {
    let peopleInIOS = ["René Schäfer", "Oliver Nowak",
                       "Jan Borchers"]

    var body: some View {
        LazyVStack(spacing: 10) {
            ForEach(peopleInIOS, id: \.self){ person in
                Text(person)
                    .font(.largeTitle)
            }
        }
    }
}
```



# Customization

# Custom Views

- Create complex views by splitting into smaller views
- Makes views reusable and simpler to exchange
- Keeps the code more readable and easier to understand

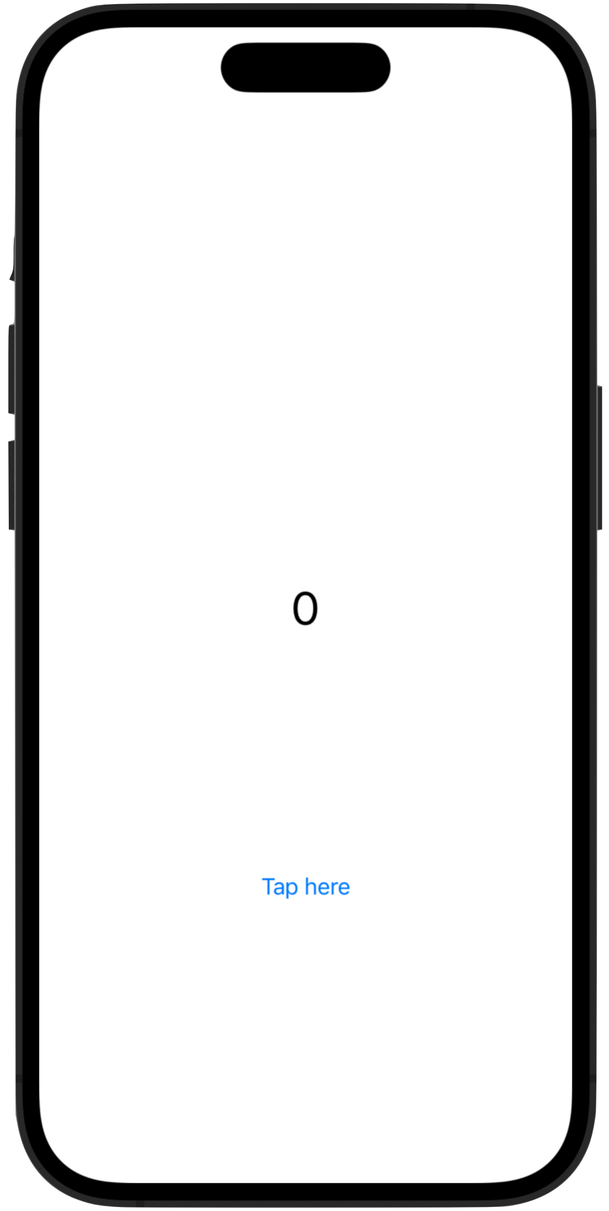
# Custom Views

```
struct ContentView: View {
    let peopleInIOS = ["René Schäfer", "Oliver Nowak", "Jan Borchers"]

    var body: some View {
        LazyVStack(spacing: 10) {
            ForEach(peopleInIOS, id: \.self){ person in
                CustomRow(name: person)
            }
        }
    }
}

struct CustomRow: View {
    var name: String

    var body: some View {
        HStack {
            Image(systemName: "checkmark")
            Text(name)
                .font(.largeTitle)
        }.padding()
    }
}
```

# Controls and Property Wrappers

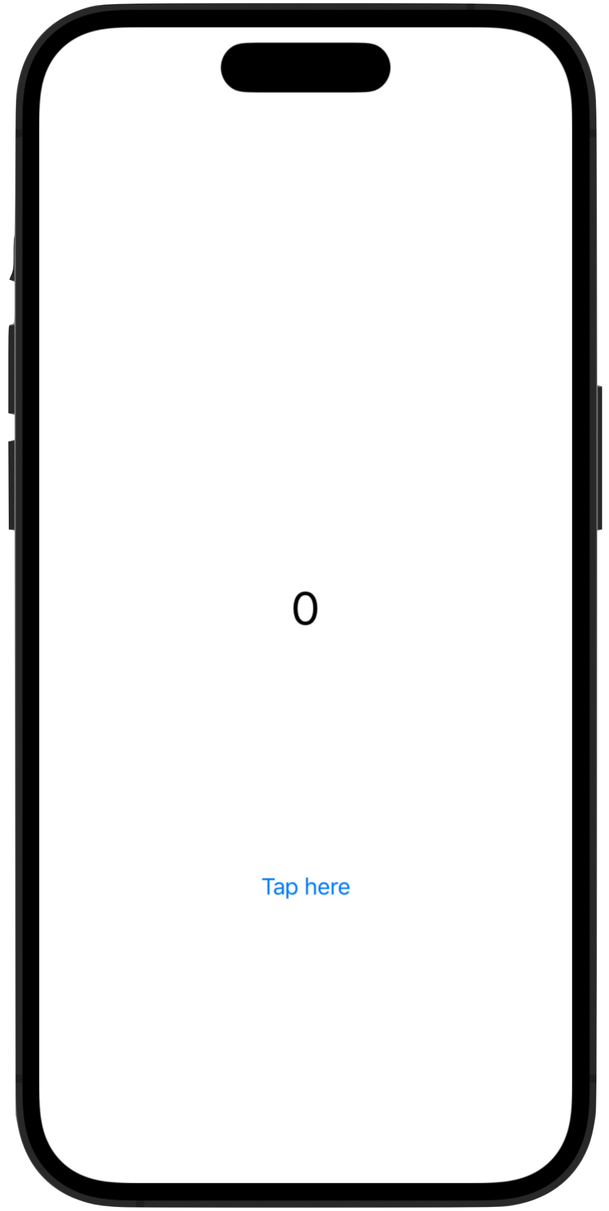
# Changing Variables

```
struct ContentView: View {
    var tapCount = 0

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\(tapCount)")
                .font(.largeTitle)
            Spacer()
            Button("Tap here") {
                tapCount += 1
            }
            Spacer()
        }

    }
}
```



# Property Wrapper: @State

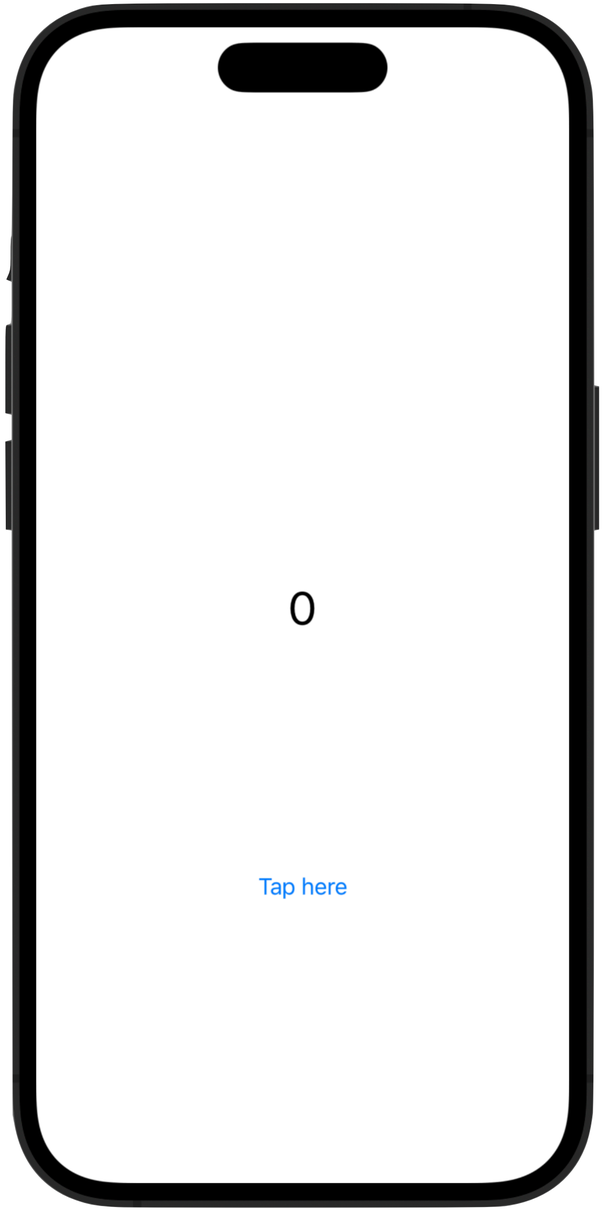
```
struct ContentView: View {
    @State private var tapCount = 0

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\(tapCount)")
                .font(.largeTitle)
            Spacer()
            Button("Tap here") {
                tapCount += 1
            }
            Spacer()
        }

    }
}
```



# Property Wrapper: @State

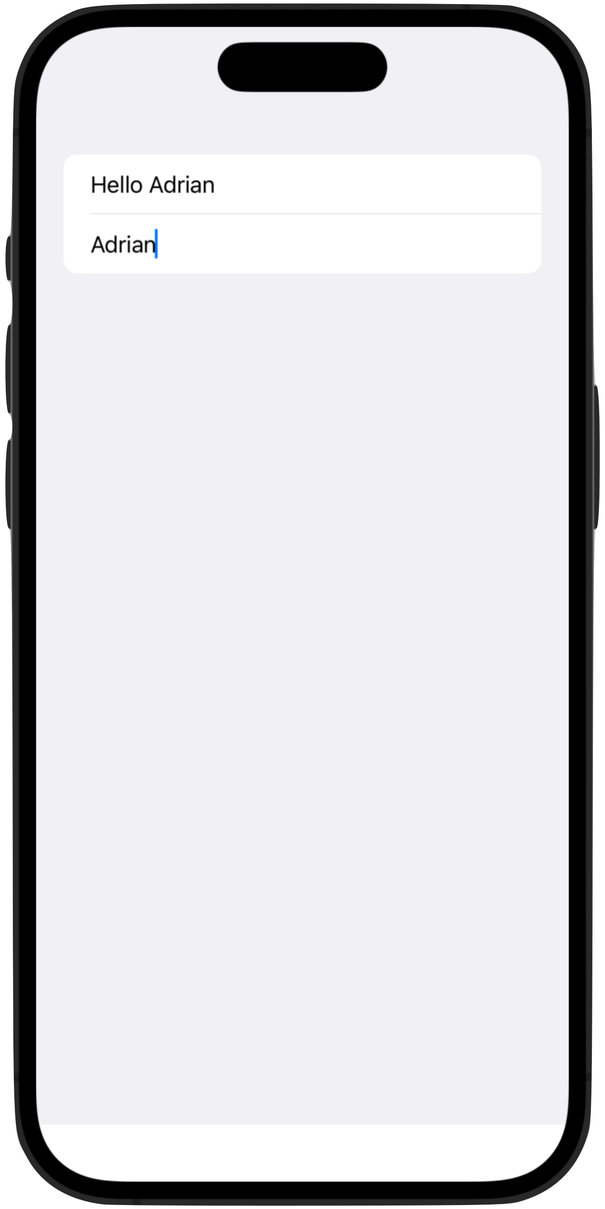
- **@State** lets you modify simple properties stored inside a single view
- Recommended to use as **private** variable
- Changing **@State** properties will reinvoke the body property
- Note: **Button**'s action is specified as trailing closure

```
struct ContentView: View {
    @State private var tapCount = 0

    var body: some View {
        VStack {
            Spacer()
            Spacer()
            Text("\(tapCount)")
                .font(.largeTitle)
            Spacer()
            Button("Tap here") {
                tapCount += 1
            }
            Spacer()
        }

    }
}
```



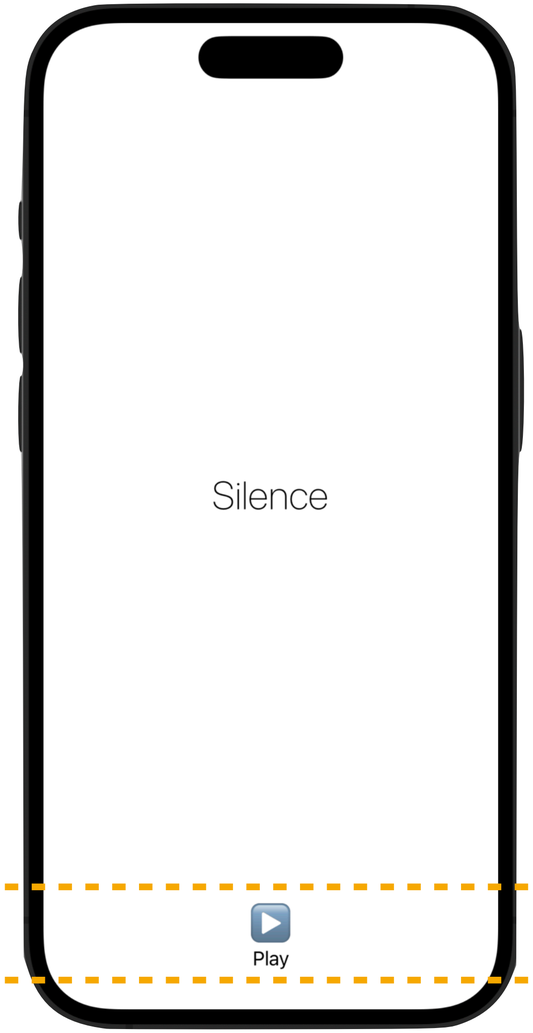
# Two-Way Bindings

```
struct ContentView: View {
    @State private var name = ""

    var body: some View {
        Form {
            Text("Hello \(name)")
            TextField("Enter your name", text: $name)
        }
    }
}
```




- **$** makes **name** a property with two-way binding: The TextField View displays its current value, but can also *change* it

# Property Wrapper: @Binding





**CustomMediaControl**

# Property Wrapper: @Binding

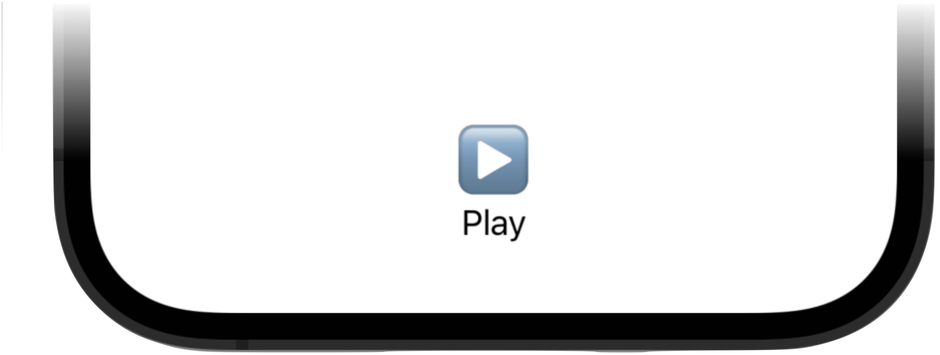
```
struct ContentView: View {
    @State private var isPlaying = false

    var body: some View {
        VStack {
            Spacer()
            if isPlaying {
                VStack {
                    Image("playImage")
                        .resizable()
                        .scaledToFit()
                    Text("Photo by Shelagh Murphy from Pexels")
                        .font(.caption)
                }
            }
            else {
                Text("Silence")
                    .font(.largeTitle)
                    .fontWeight(.thin)
            }
            Spacer()
            CustomMediaControl(isPlaying: $isPlaying)
        }
        .frame(alignment: .bottom)
    }
}
```

# Property Wrapper: @Binding

```
struct CustomMediaControl: View {
    @State var isPlaying: Bool

    var body: some View {
        VStack{
            Button(isPlaying ? "⏸️" : "▶️") {
                isPlaying.toggle()
            }
            .font(.largeTitle)
            Text(isPlaying ? "Enjoy the silence" : "Play")
        }
    }
}
```

# Property Wrapper: @Binding

```
struct CustomMediaControl: View {
    @Binding var isPlaying: Bool

    var body: some View {
        VStack{
            Button(isPlaying ? "⏸" : "▶️") {
                isPlaying.toggle()
            }
            .font(.largeTitle)
            Text(isPlaying ? "Enjoy the silence" : "Play")
        }
    }
}
```



# Other Property Wrappers

- `@StateObject`
  A property wrapper type that instantiates an observable object.

- `@ObservedObject`
  A property wrapper type that subscribes to an observable object and invalidates a view whenever the observable object changes.

- `@EnvironmentObject`
  A property wrapper type for an observable object supplied by a parent or ancestor view.

- `@Environment`
  A property wrapper that reads a value from a view’s environment.

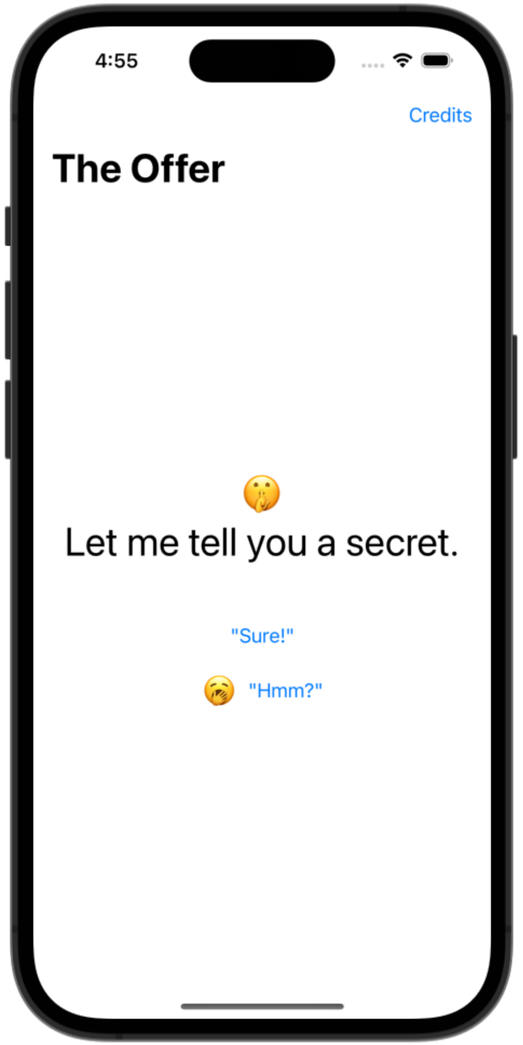
# Navigation

# Drill-Down Navigation Example

# Using NavigationStack

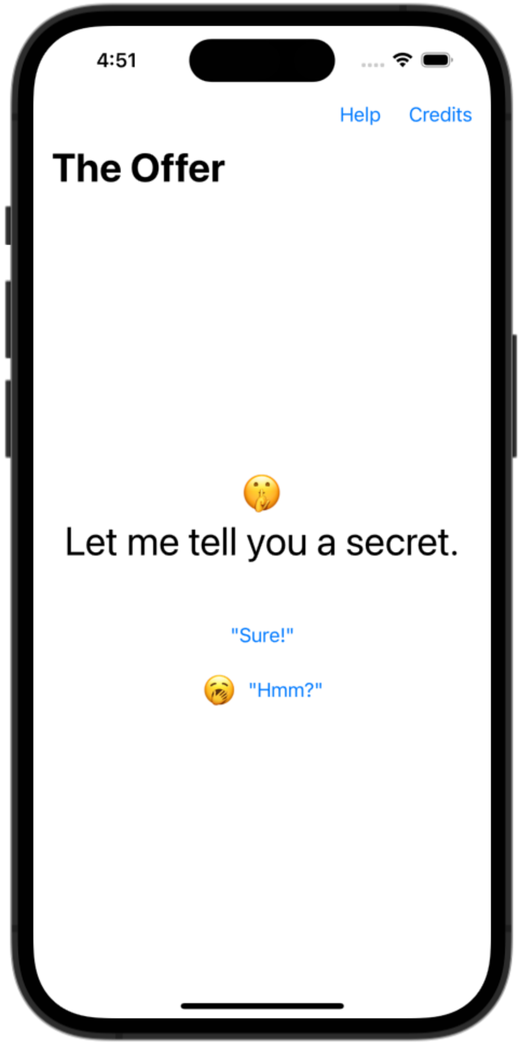
```
struct ContentView: View {
    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("🤫\nLet me tell you a secret.")
                    .padding(.bottom, 30.0)
                    .font(.largeTitle).multilineTextAlignment(.center)
                NavigationLink("\"Sure!\"", destination: PositiveAnswerView())
                NavigationLink(destination: NegativeAnswerView()) {
                    HStack {
                        Text("😮‍💨")
                            .font(.title)
                        Text("\"Hmm?\"")
                    }
                }
            }
            .navigationTitle(Text("The Offer"))
            .toolbar{
                ToolbarItem(placement: .navigationBarTrailing){
                    Button("Credits"){}
                }
            }
        }
    }
}
```

← Added to the VStack!



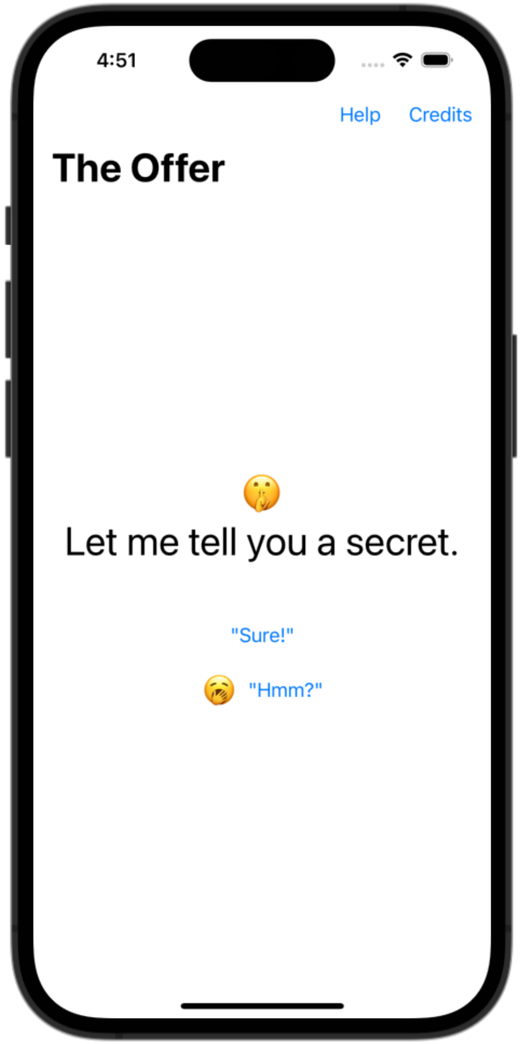
# NavigationStack: Grouping Toolbar Items

```
struct ContentView: View {
    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("🤫\nLet me tell you a secret.")
                    .padding(.bottom, 30.0)
                    .font(.largeTitle).multilineTextAlignment(.center)
                NavigationLink("\"Sure!\"", destination: PositiveAnswerView())
                NavigationLink(destination: NegativeAnswerView()) {
                    HStack {
                        Text("😒")
                            .font(.title)
                        Text("\"Hmm?\"")
                    }
                }
            }
            .navigationTitle(Text("The Offer"))
            .toolbar{
                ToolbarItemGroup(placement: .navigationBarTrailing){
                    Button(“Credits"){}
                    Button(“Help"){}
                }
            }
        }
    }
}
```



# NavigationStack: NavigationLinks

```
struct ContentView: View {
    var body: some View {
        NavigationStack {
            VStack(spacing: 20) {
                Text("🤫\nLet me tell you a secret.")
                    .padding(.bottom, 30.0)
                    .font(.largeTitle).multilineTextAlignment(.center)
                NavigationLink("\"Sure!\"", destination: PositiveAnswerView())
                NavigationLink(destination: NegativeAnswerView()) {
                    HStack {
                        Text("😮‍💨")
                            .font(.title)
                        Text("\"Hmm?\"")
                    }
                }
            }
            .navigationTitle(Text("The Offer"))
            .toolbar{
                ToolbarItemGroup(placement: .navigationBarTrailing){
                    Button(“Credits"){}
                    Button(“Help"){}
                }
            }
        }
    }
}
```

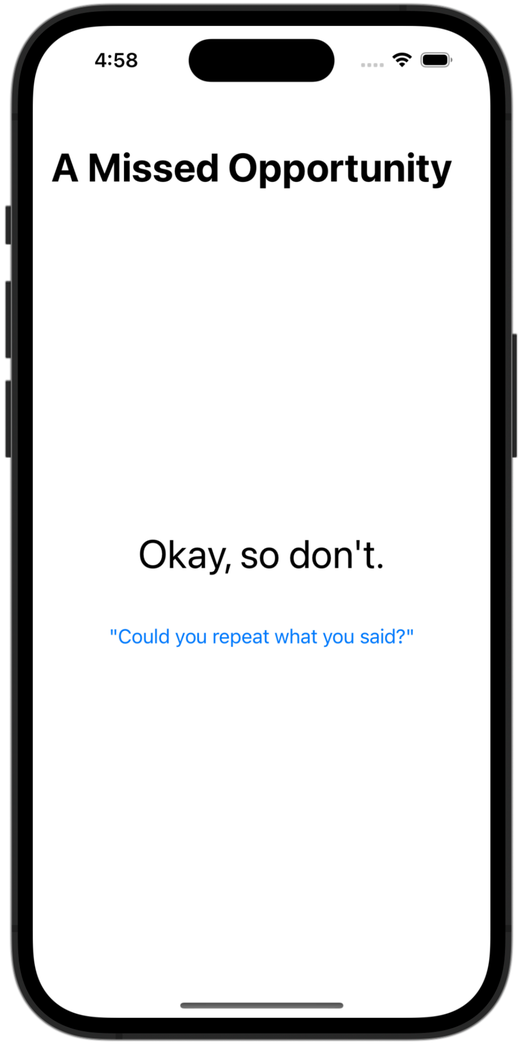


# A Simple Destination View

```
struct PositiveAnswerView: View {
    var body: some View {
        VStack() {
            Text("SwiftUI is awesome! 🎉")
                .font(.largeTitle)
        }
        .navigationTitle(Text("The Secret"))
    }
}
```

# Custom Destination View: @Environment

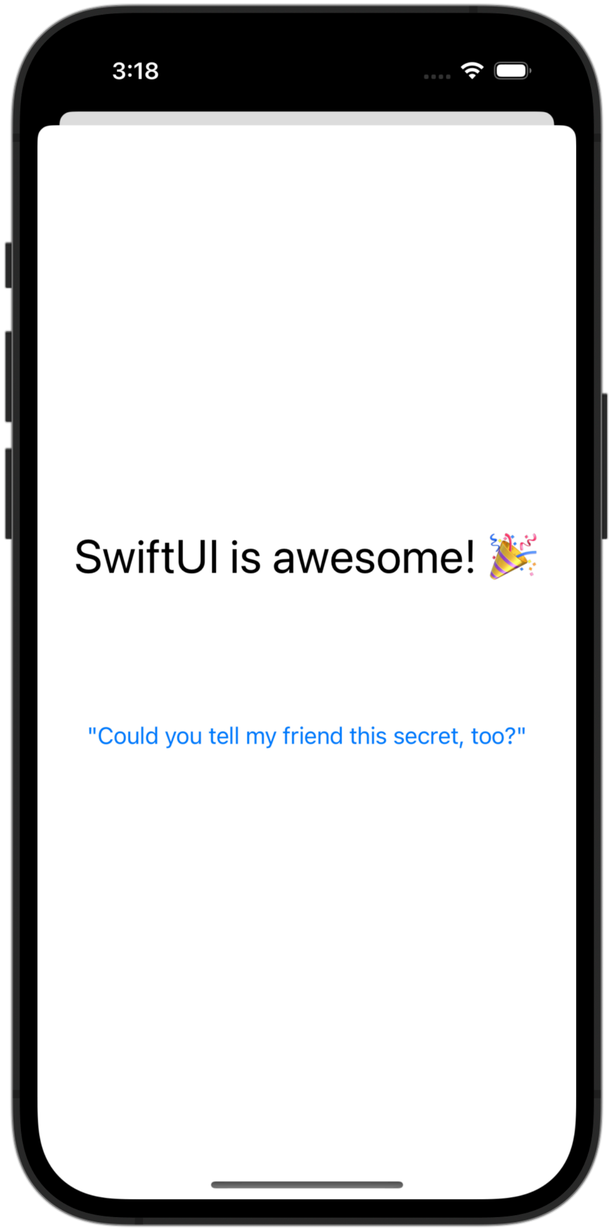
```
struct NegativeAnswerView: View {
    @Environment(\.presentationMode) var presentationMode

    var body: some View {
        VStack(spacing: 40) {
            Text("Okay, so don't.")
                .font(.largeTitle)
            Button("\"Could you repeat what you said?\"", action: {
                presentationMode.wrappedValue.dismiss()
            })
        }
        .navigationTitle(Text("A Missed Opportunity"))
        .navigationBarBackButtonHidden(true)
    }
}
```



# Sheets & How To Pass Data

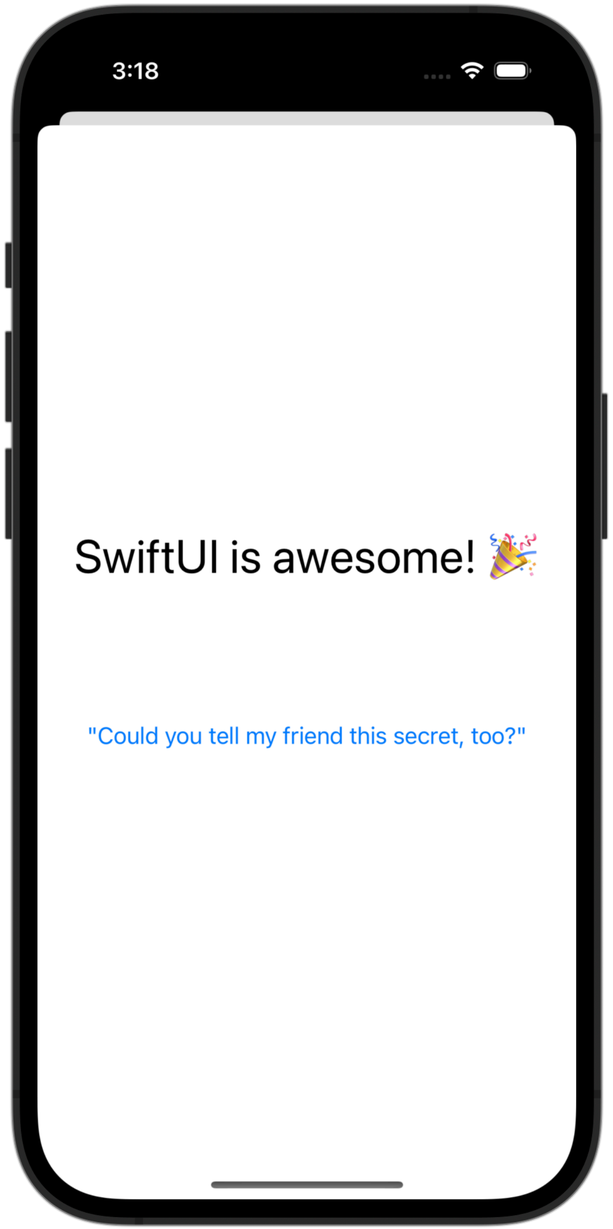
```
struct ContentView: View {

    @State var showSecret = false

    var body: some View {
       VStack(spacing: 20) {
            Text("🤫\nLet me tell you a secret.")
                .padding(.bottom, 30.0)
                .font(.largeTitle).multilineTextAlignment(.center)
            Button("\"Sure!\""){
                showSecret.toggle()
            }
            .sheet(isPresented: $showSecret){
                PositiveAnswerView(showSecret: $showSecret)
            }
        }
    }
}
```

Also works this way with NavigationLinks



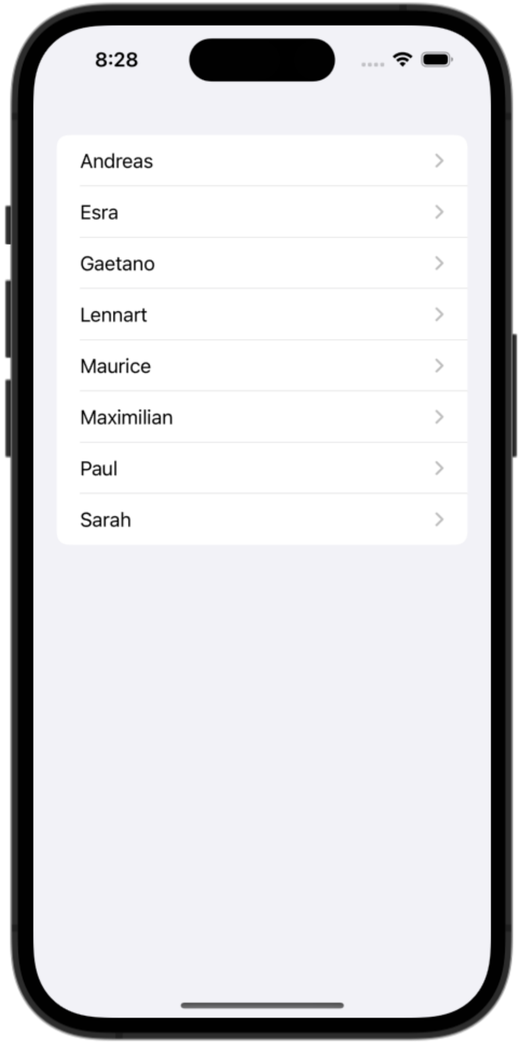
# Sheets & How To Pass Data

```
struct PositiveAnswerView: View {
    @Binding var showSecret: Bool

    var body: some View {
        VStack(spacing: 100) {
            Text("SwiftUI is awesome! 🎉")
                .font(.largeTitle)
            Button("\"Could you tell my friend this secret, too?\""){
                showSecret = false
            }
        }
    }
}
```



# NavigationLinks: NavigationDestination

```
struct ContentView: View {
    let people = ["Andreas", "Esra", "Gaetano", "Lennart", "Maurice",
                  "Maximilian", "Paul", "Sarah"]

    var body: some View {
        NavigationStack {
            List(people, id: \.self){ person in
                NavigationLink(person, value: person)
            }
            .navigationDestination(for: String.self){ person in
                ProfilePage(person: person)
            }
        }
    }
}
```



# SwiftUI: Further Reading

- Apple's SwiftUI Tutorials, and WWDC presentations
- Paul Hegarty, Stanford University: CS193p — Developing Applications for iOS
- Paul Hudson: 100 Days of SwiftUI (free), Swift Design Patterns ($25–50)
- Mark Moeykens: SwiftUI View Mastery
- Next release of Apple's free Develop in Swift books (currently still UIKit only)

# SwiftUI: the Big Messages

1. Object-Oriented Programming is Dead, Long Live **Declarative Programming!**
2. **MVVM** is the corresponding modern improvement over MVC
3. Modern universal languages can describe UIs like **domain-specific languages**
4. You can design a UI **graphically and in code simultaneously**
5. The best app languages must **evolve together** with a UI library and IDE
6. Declarative Programming simplifies development across **mobile and desktop**
7. SwiftUI is a current **case study of a paradigm shift** across a major OS family