Any questions, comments, or concerns from last week?

We value your feedback!

(please)

SwiftUI Fundamentals

Lecture 3

CIS 1951

Last time, in CIS 1951…

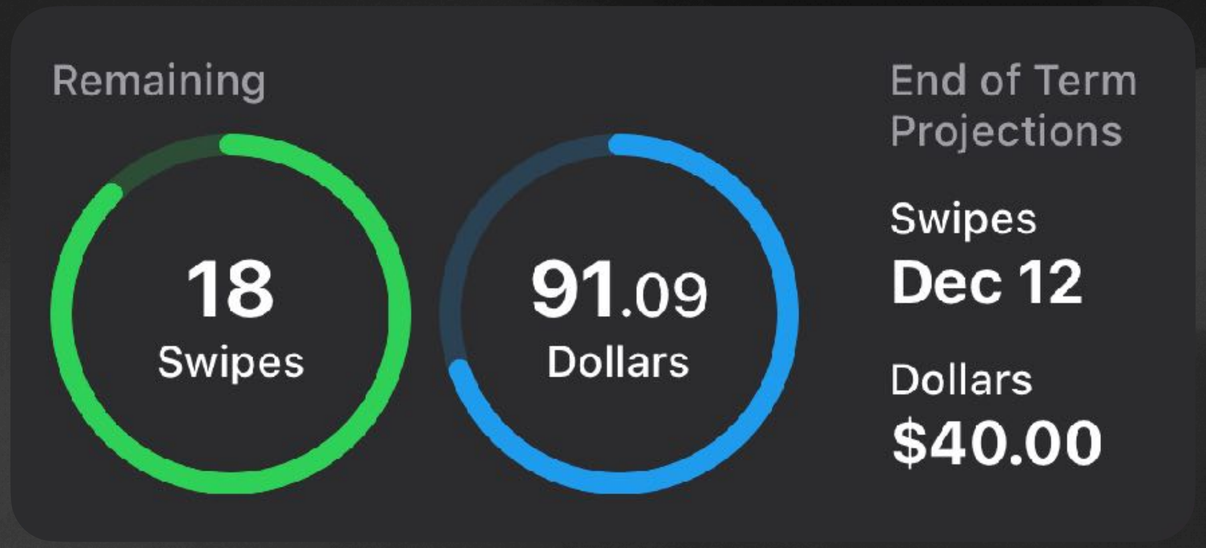
How to Swift!

- Declaring constants and variables is easy with **type inference**
- **Optionals** make handling nil/null much safer
- **Functions and closures** are first-class types
- **Classes, structs, and enums** let us organize code and data
- **Protocols and extensions** make defining and standardizing behavior easy
- **Questions? Comments?**

Swift

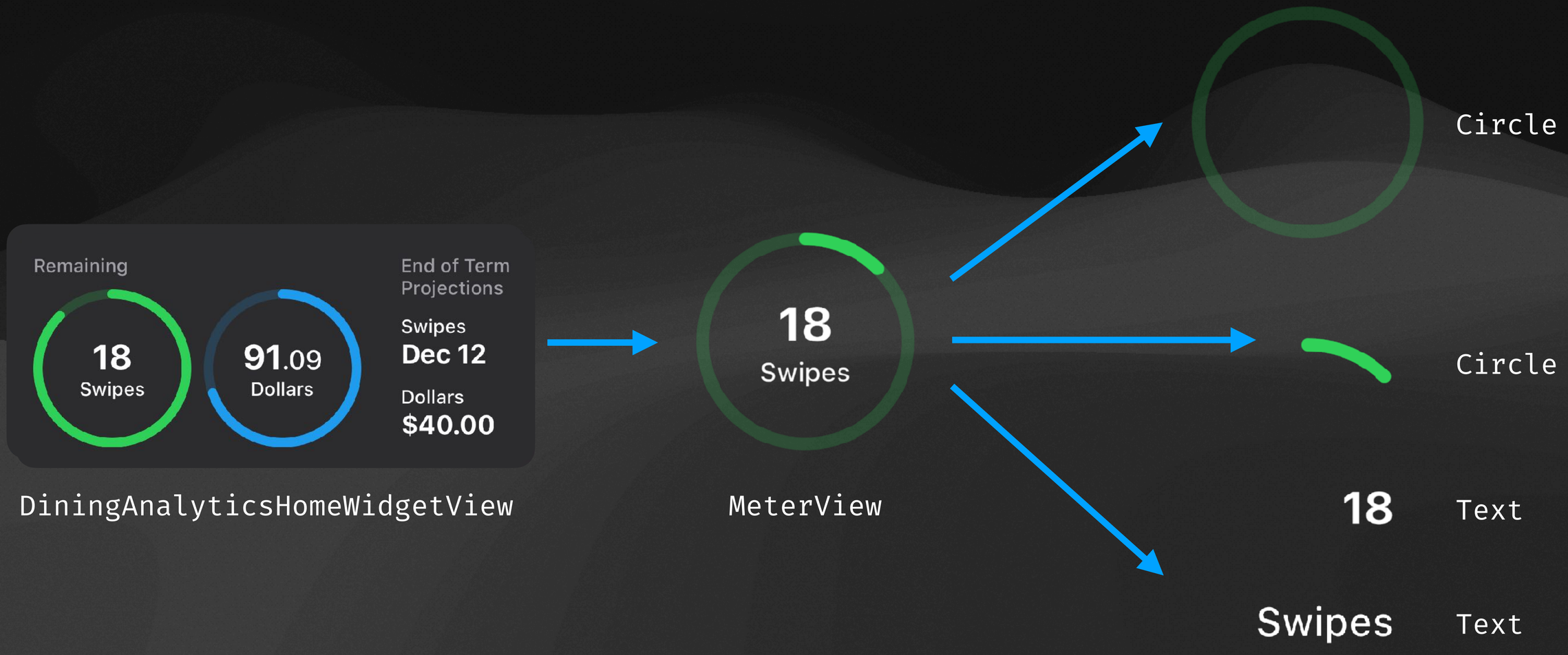
SwiftUI

Views are the **building blocks** of UI



`DiningAnalyticsHomeWidgetView`

Views are composed of **other views**

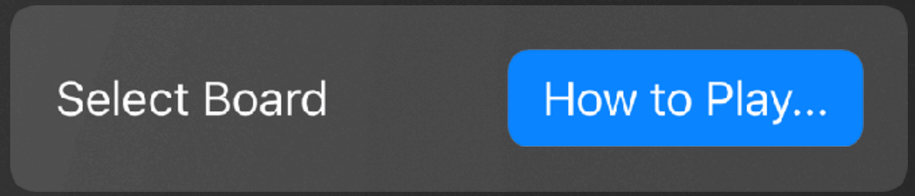


Views are **Swift structs**

```
public protocol View {
    /// The type of view representing the body of this view.
    associatedtype Body : View

    /// The content and behavior of the view.
    var body: Self.Body { get }
}
```

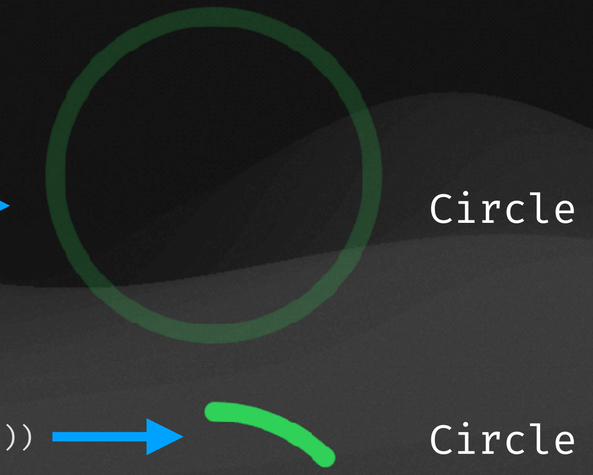
Views are **Swift structs**

```
struct HeaderView: View {
    var body: some View {
        HStack {
            Text("Select Board")
            Spacer()
            Button("How to Play ... ") {}
                .buttonStyle(.borderedProminent)
        }
    }
}
```



Views are **functions of their data**

```
struct MeterView: View {
    var current: Double
    other properties ...

    var body: some View {
        ZStack {
            Circle()
                .stroke(color.opacity(0.2), lineWidth: lineWidth)
                .aspectRatio(1, contentMode: .fit)

            Circle()
                .trim(from: 0, to: max(0, min(current / maximum, 1)))
                .rotation(.degrees(-90))
                .stroke(color, style: StrokeStyle(lineWidth: lineWidth, lineCap: .round))
                .aspectRatio(1, contentMode: .fit)

            VStack {
                Text("\(Int(current))")
                    .fontWeight(.bold)
                    .font(.title2)

                Text(description)
                    .fontWeight(.medium)
                    .font(.caption)
            }
        }
    }
}
```



Primitive Views

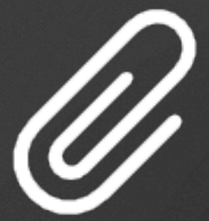
Text

```
Text("Hello! I am text!")
```

Primitive Views

Image

```
Image(systemName: "paperclip")
```

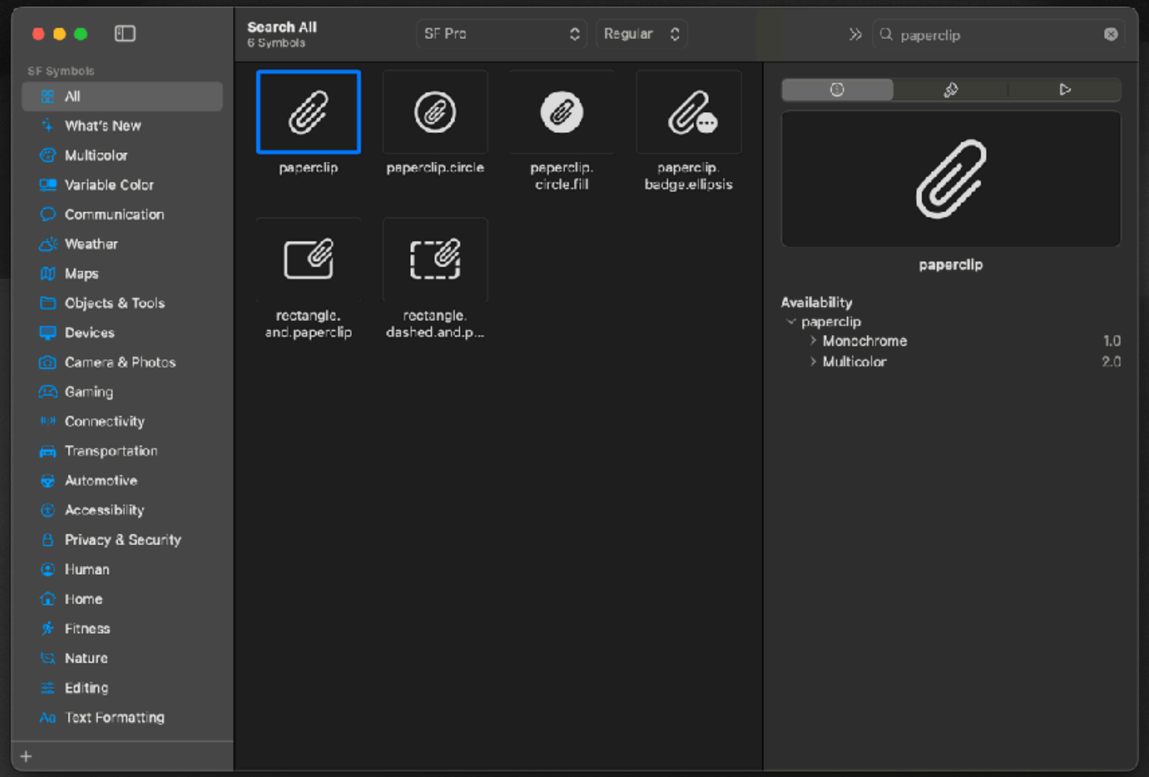


```
Image("Landscape_4")
```

Primitive Views

Image

```
Image(systemName: "paperclip")
```

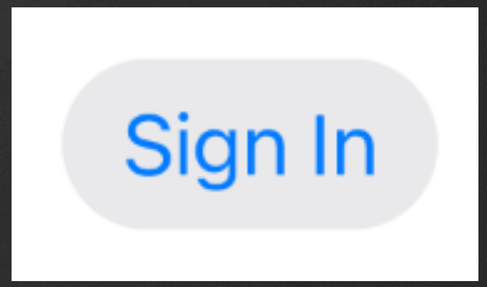


SF Symbols

Primitive Views

Button

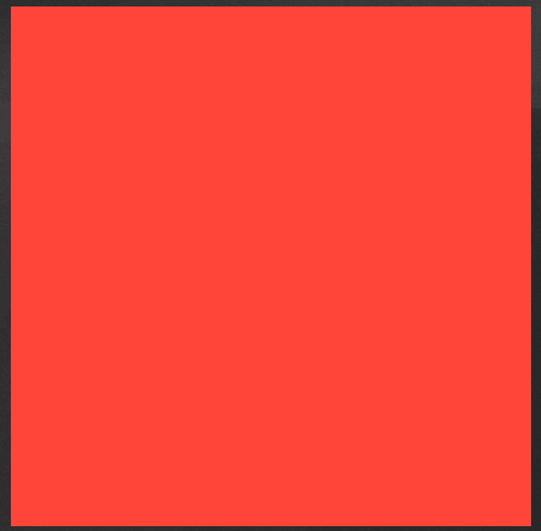
```
Button(action: signIn) {
    Text("Sign In")
}
```



Primitive Views

Color

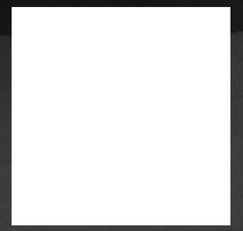
```
Color.red
```



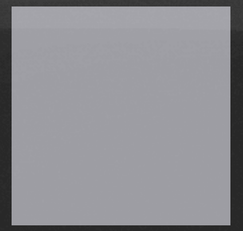
Primitive Views

Color

`Color.primary`



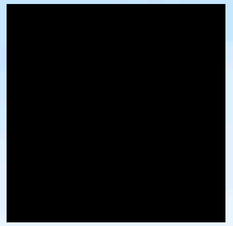
`Color.secondary`



Primitive Views

Color

Color.primary



Color.secondary

Primitive Views

Stacks

```
HStack {
    Text("1")
    Text("2")
    Text("3")
}
```

1 2 3

```
VStack {
    Text("1")
    Text("2")
    Text("3")
}
```

1

2

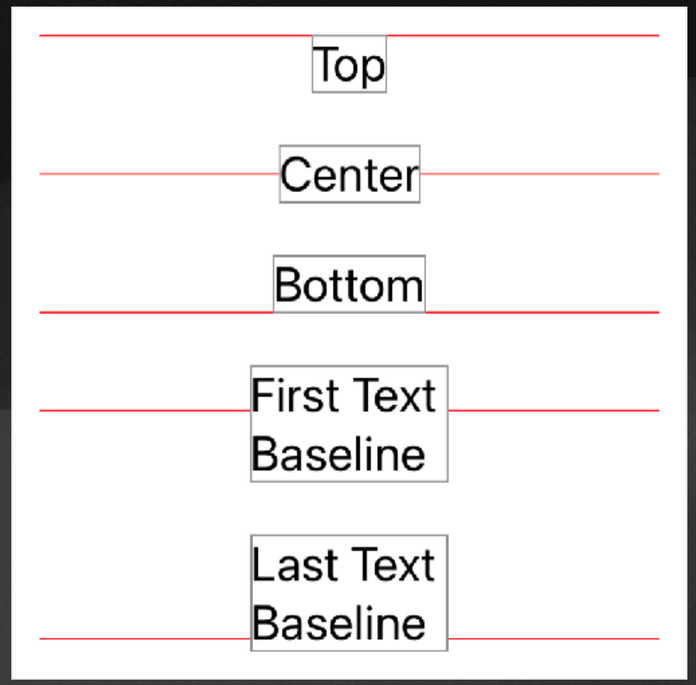
3

Primitive Views

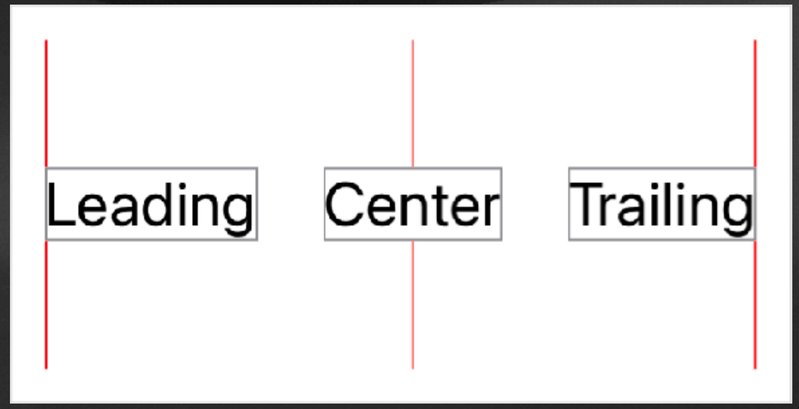
Stack Alignment & Spacing

```
var body: some View {
    HStack(alignment: .top, spacing: 10) {
        ForEach(1...5, id: \.self) {
            Text("Item \($0)")
        }
    }
}
```

Aligning Elements



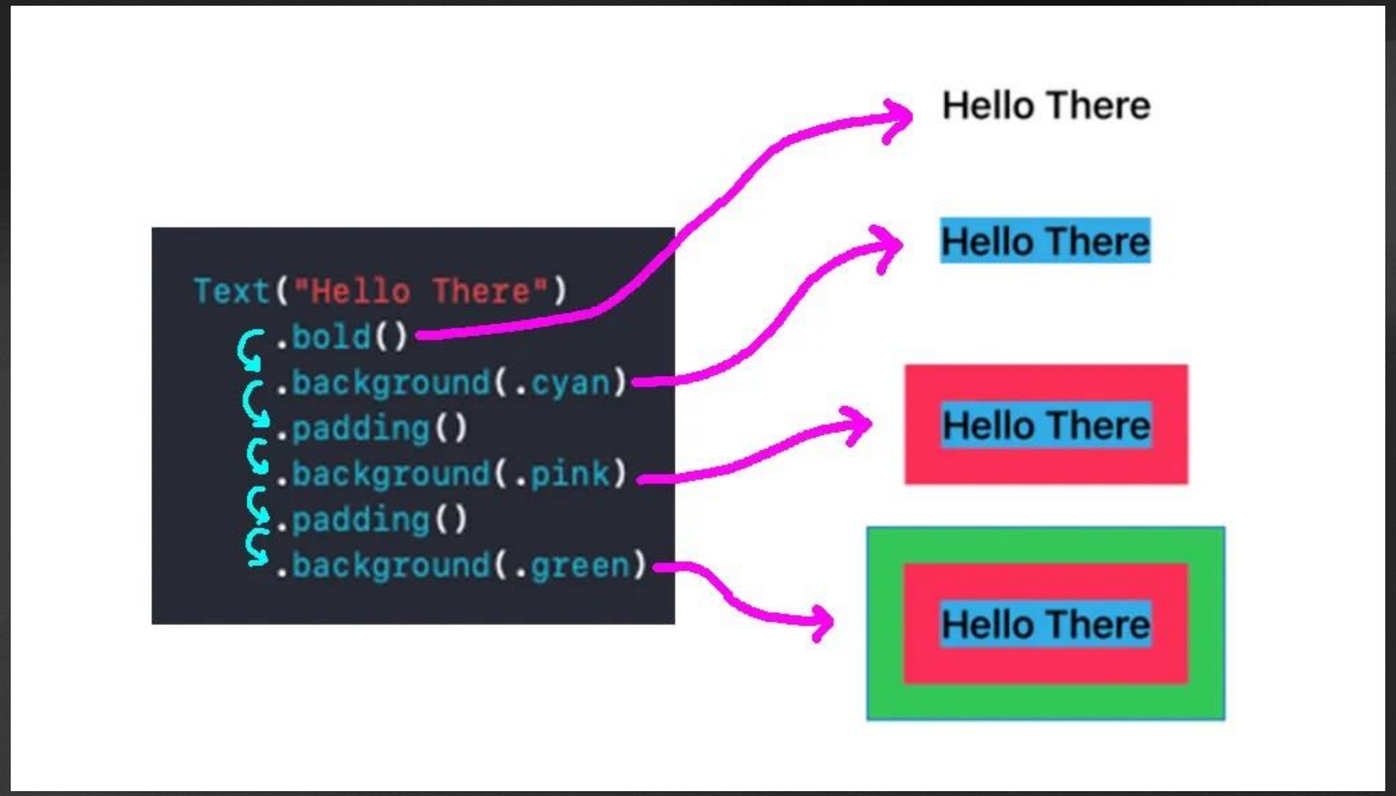
Vertical Alignment



Horizontal Alignment

Adding Modifiers

A **view modifier** is a protocol method applied to a view or another view modifier to **produce a new version** of the original value



“Example of view modifiers chain”, diagram by Ricardo Montemayor

Common Modifiers

```
Text("Important!")
    .foregroundStyle(.yellow)
```

Common Modifiers

```
Text("Important!")
    .foregroundStyle(.yellow)
    .padding()
```

Common Modifiers

```
Text("Important!")
    .foregroundStyle(.yellow)
    .frame(width: 100, height: 100)
```

Common Modifiers

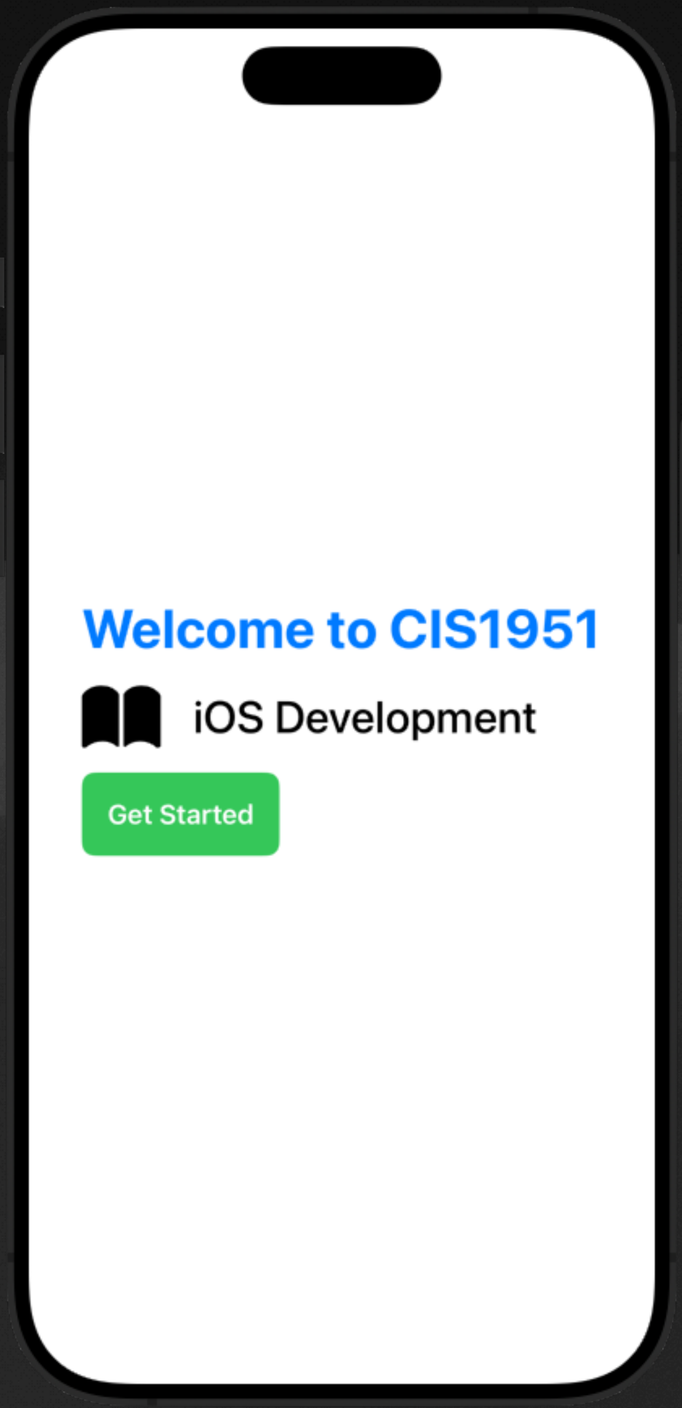
```
Text("Important!")
    .foregroundStyle(.yellow)
    .frame(width: 100, height: 100)
    .background(.red)
```

Live Demo

Test Your Knowledge

- Can you code up this UI?

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack(alignment: .leading, spacing: 10) {
            Text("Welcome to CIS1951")
                .font(.largeTitle)
                .fontWeight(.bold)
                .foregroundColor(.blue)

            HStack {
                Image(systemName: "book.fill")
                    .resizable()
                    .aspectRatio(contentMode: .fit)
                    .frame(width: 50, height: 50)
                Spacer()
                    .frame(width: 20)
                Text("iOS Development")
                    .font(.title)
                    .fontWeight(.medium)
            }

            Button(action: {}) {
                Text("Get Started")
                    .font(.headline)
                    .padding()
                    .background(Color.green)
                    .foregroundColor(.white)
                    .cornerRadius(8)
            }
        }
    }
}
```



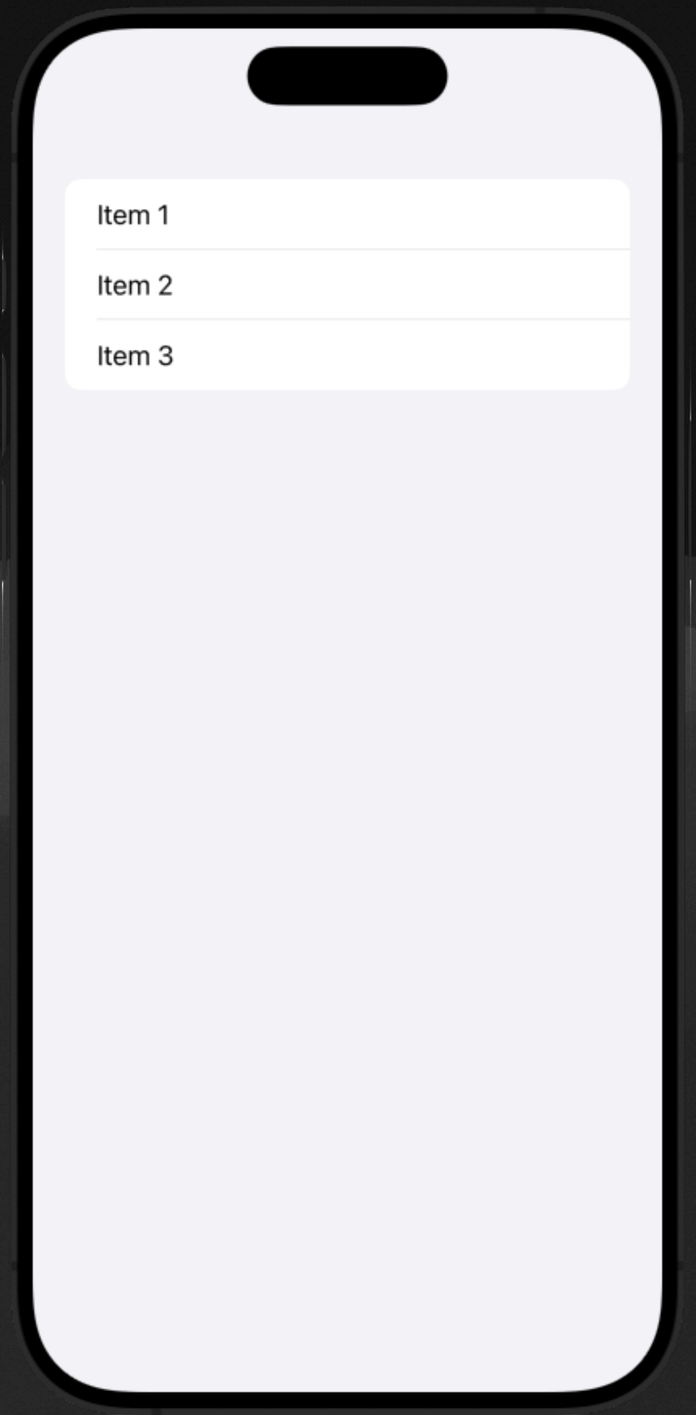
List and ScrollView: Scrollable UI

- **List** presents vertically scrollable data in a single column
- **ScrollView** can scroll content horizontally, vertically, or both
  - Often used with VStack and ForEach
- List does **lazy load**, while ScrollView loads everything at once
  - ScrollView with large data: use LazyVStack and LazyHStack

List

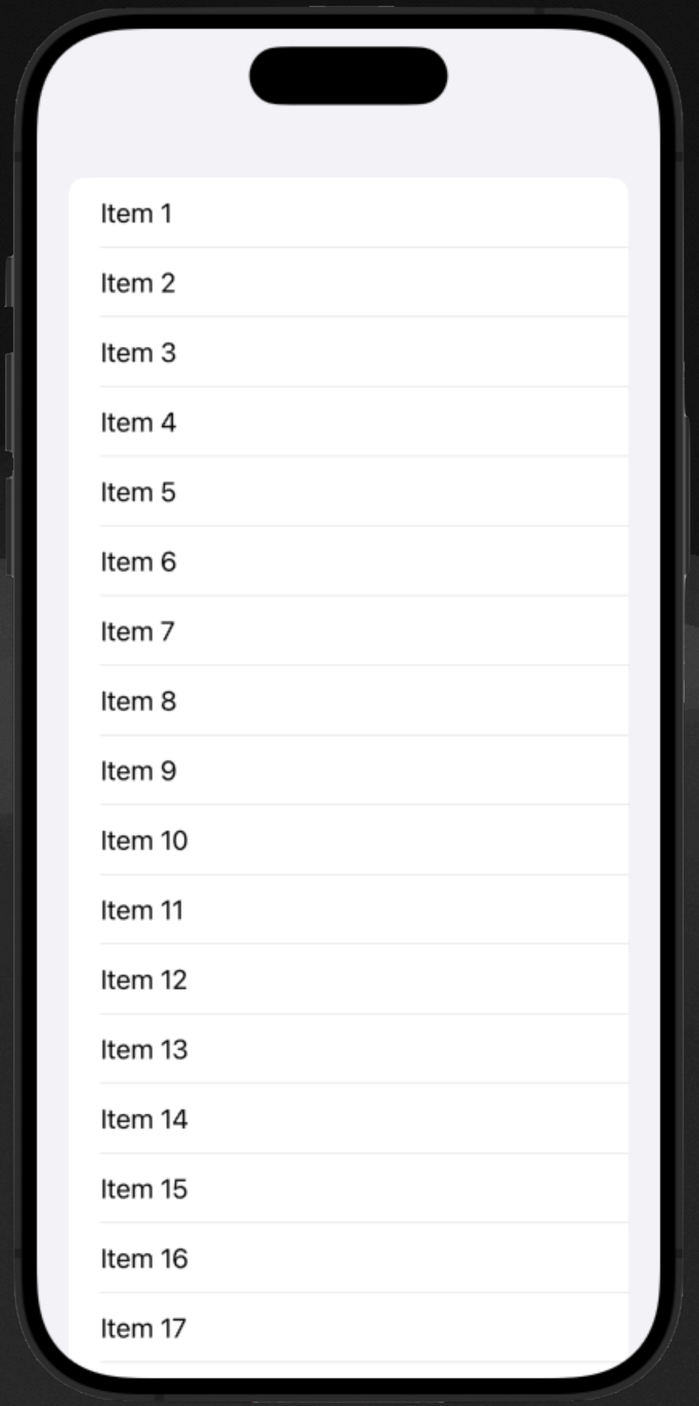
```
struct ContentView: View {
    let items = ["Item 1", "Item 2", "Item 3"]

    var body: some View {
        List(items, id: \.self) { item in
            Text(item)
        }
    }
}
```

Needed if items don’t conform to Identifiable



List

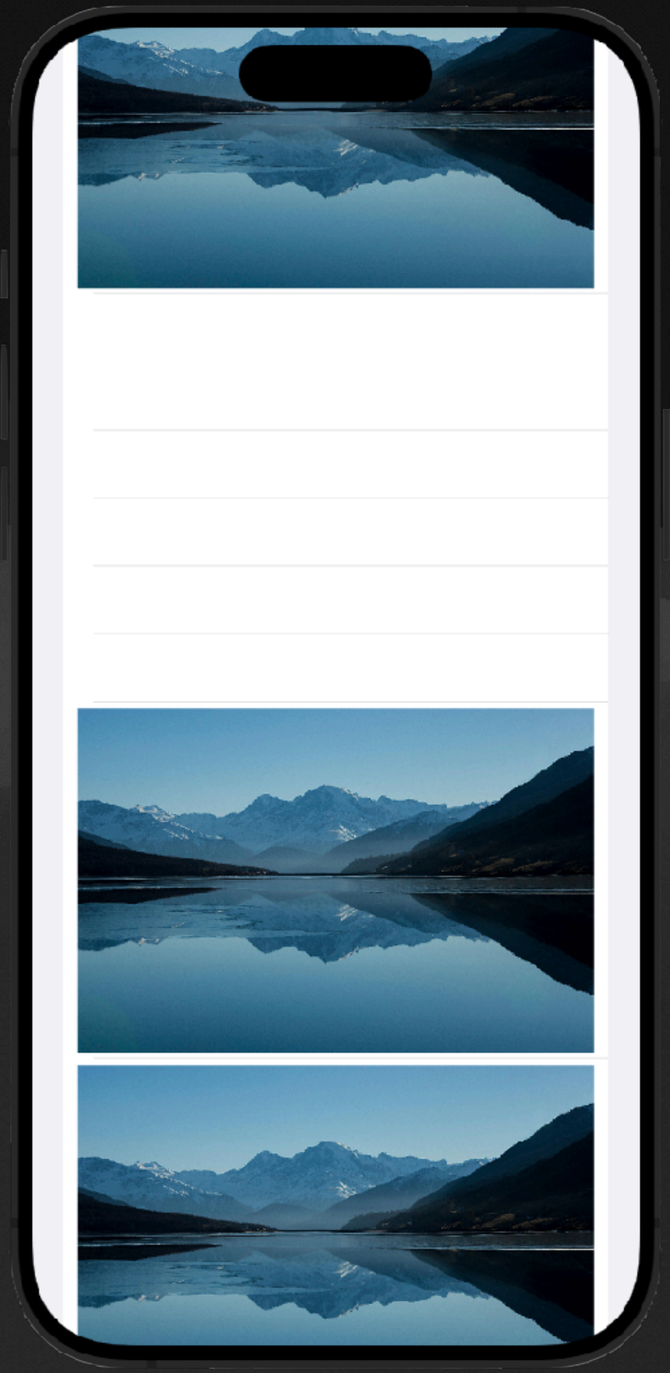
```
struct ContentView: View {
    var body: some View {
        List {
            ForEach(1..<100) { index in
                Text("Item \(index)")
            }
        }
    }
}
```



List

- When you scroll too fast and your item loads too slowly… Lazy load is exposed

```
struct ContentView: View {
    var body: some View {
        List {
            ForEach(1..<100) { index in
                AsyncImage(url: URL(string:
  "https://images.pexels.com/photos/346529/pexels-photo-346529.jpeg?
  cs=srgb&dl=pexels-bri-schneiter-28802-346529.jpg&fm=jpg")) { result in
                    result.image?
                        .resizable()
                        .scaledToFill()
                }
            }
        }
    }
}
```

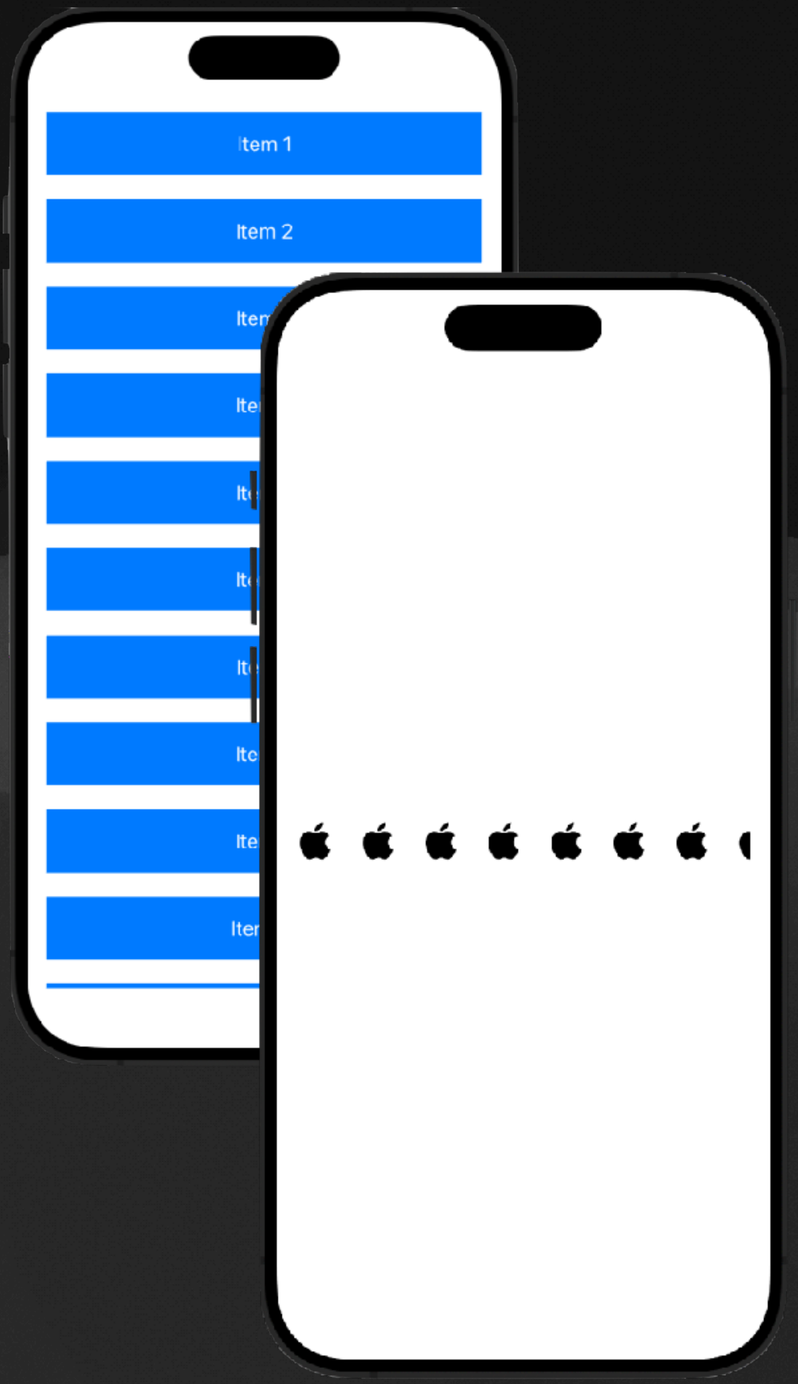


ScrollView

- Vertical and horizontal scrolls
- Modifiers to customize the UI

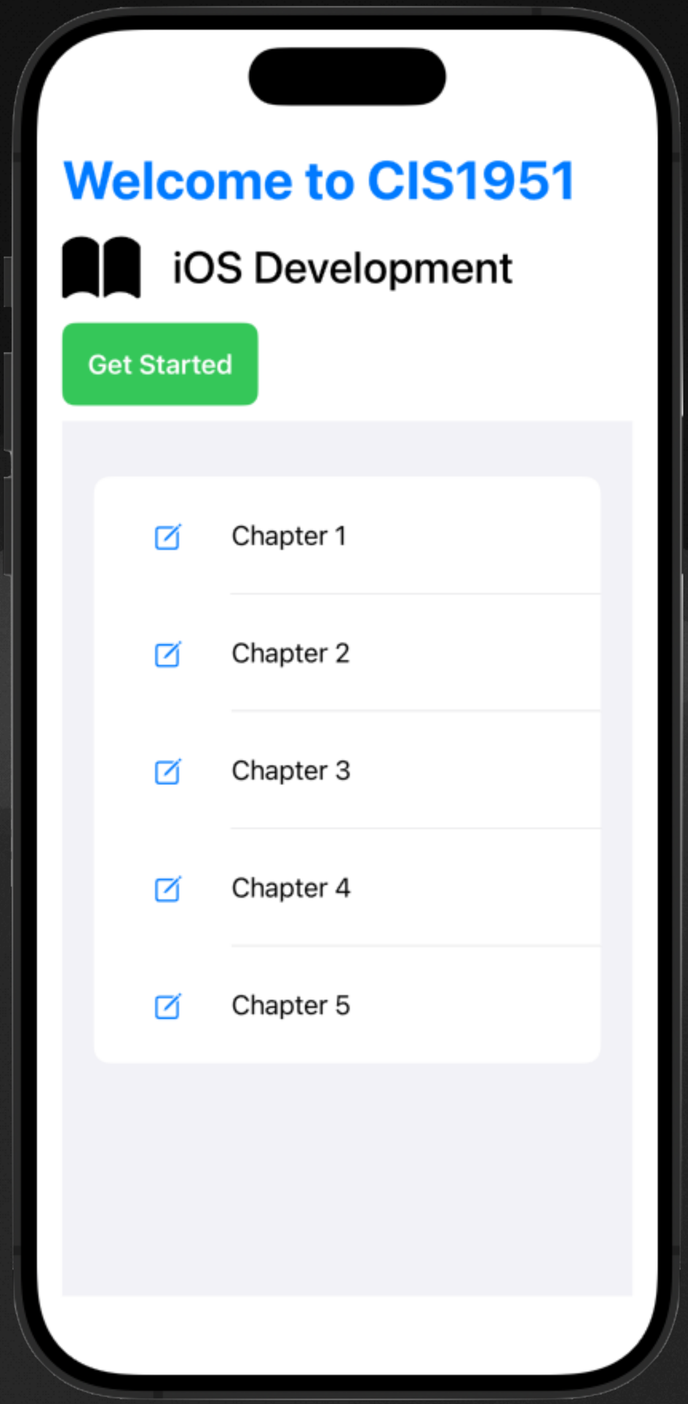
```swift
// Vertical Scroll View
ScrollView {
    VStack(spacing: 20) {
        ForEach(1..<20) { i in
            Text("Item \(i)")
                .frame(maxWidth: .infinity)
                .padding()
                .background(Color.blue)
                .foregroundColor(.white)
        }
    }
}
.padding()
.scrollIndicators(.hidden)
```

```swift
// Horizontal Scroll View
ScrollView(.horizontal) {
    HStack(spacing: 20) {
        ForEach(1..<10) { i in
            Image(systemName: "apple.logo")
                .resizable()
                .aspectRatio(contentMode: .fit)
                .frame(width: 30, height: 30)
        }
    }
}
.padding()
.scrollIndicators(.hidden)
```



Test Your Knowledge

- Can you code up this UI?



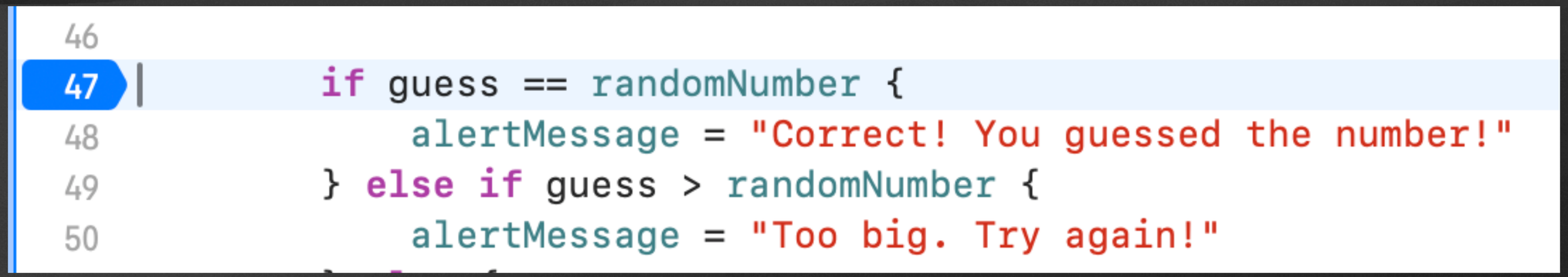
Your goal: make Sidechat

Amazing! You can now build with SwiftUI.

Next, what to do if it goes wrong?

Debugging in Xcode: Breakpoints

- A **breakpoint** is a marker in code that pauses an app during execution. It allows us to inspect the state of the app at that point



Today, we learned…

- What is SwiftUI
- How to SwiftUI
  - Views, layouts, modifiers, lists, scrolling views
- Xcode debugging tools

Next time…

- Week 4: SwiftUI state management (add data to views!)
- HW1 is due Wednesday, 2/12 @ 11:59PM