iOS Bootcamp - Meeting 3

Hosted by App Team Carolina

Agenda

What can you expect this meeting?

1. **Announcements:**
   a. Next meeting is **Oct. 14th** in **Gardner 008**
2. Recap
3. @State
4. @Binding
5. Countey Demo

Attendance!

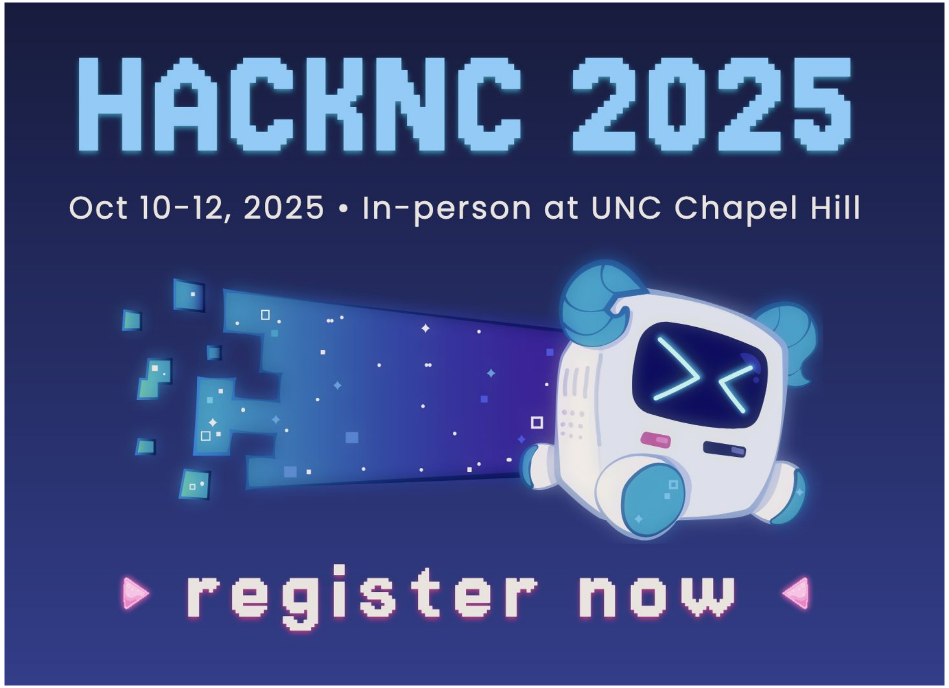
Please fill this out!

HackNC

Oct 10-12

Sign-up!

- Very fun
- Learning opportunity
- Career opportunity
- **Make a SwiftUI app!**
- Register by Friday



Recap

Recap

What did you learn last meeting?

1. Structs
   - Represent data
2. Subviews
   - Reuse code
3. ForEach
   - Dynamically generate subviews

ForEach

What is ID?

ID is:

- The property that **uniquely identifies** each element
- Needed to add and remove elements from a collection

```swift
struct ForEachView: View {
    let names = ["Hussain", "Alex", "Tri"]

    var body: some View {
        ForEach(names, id: \.self) { name in
            Text(name)
        }
    }
}
```

id: \.self means an element is using its own value as a unique identifier

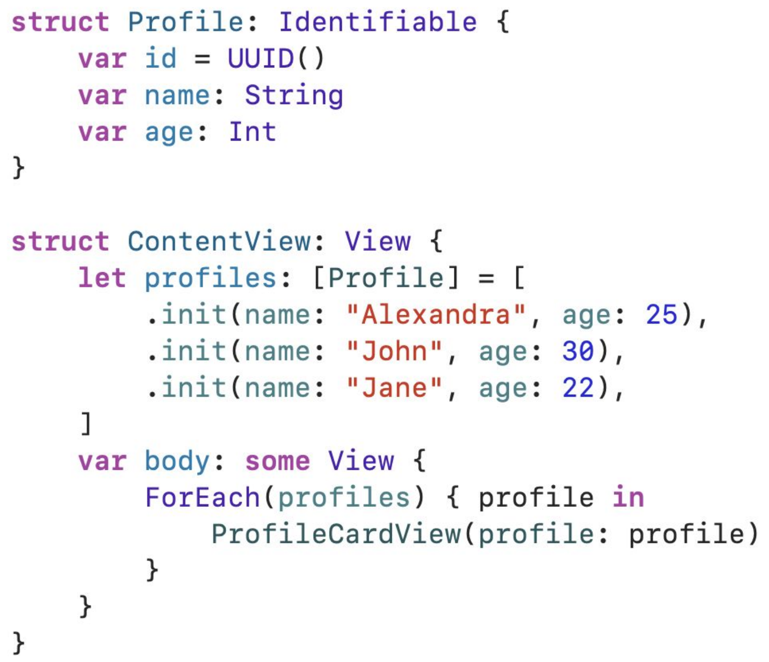
ForEach

What is ID?

A custom type's ID can be:

- A struct property (i.e. **id: \.name**)
- Conform to identifiable (i.e. **id: \.id**)

We prefer to use **identifiable**. It makes the ForEach more readable and guarantees uniqueness.



```swift
struct Profile: Identifiable {
    var id = UUID()
    var name: String
    var age: Int
}

struct ContentView: View {
    let profiles: [Profile] = [
        .init(name: "Alexandra", age: 25),
        .init(name: "John", age: 30),
        .init(name: "Jane", age: 22),
    ]
    var body: some View {
        ForEach(profiles) { profile in
            ProfileCardView(profile: profile)
        }
    }
}
```

ForEach

What is ID?

```swift
struct Profile: Identifiable {
    var id = UUID()
    var name: String
    var age: Int
}

struct ContentView: View {
    let profiles: [Profile] = [
        .init(name: "Alexandra", age: 25),
        .init(name: "John", age: 30),
        .init(name: "Jane", age: 22),
    ]
    var body: some View {
        ForEach(profiles) { profile in
            ProfileCardView(profile: profile)
        }
    }
}
```

```swift
struct Profile {
    var name: String
    var age: Int
}

struct ContentView: View {
    let profiles: [Profile] = [
        .init(name: "Alexandra", age: 25),
        .init(name: "John", age: 30),
        .init(name: "Jane", age: 22),
    ]
    var body: some View {
        ForEach(profiles, id: \.name) { profile in
            ProfileCardView(profile: profile)
        }
    }
}
```

User Input + @State

Intro to User Input

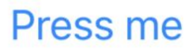
Button

So far, we’ve built good-looking UI. But how do we make these apps interactive?

SwiftUI provides a number of interactive views – **Button** is the most fundamental.

```swift
Button("Press me") {
    print("Hello, World!")
}
```

$\longrightarrow$


Press me

Intro to User Input

Button

Button has two parameters:

1. **Action** – the Swift code executed when the button is tapped.
2. **Label** – a View that determines how the button appears

```
Button {
    print("Hello, World!")
} label: {
    Text("Press me")
        .padding()
        .foregroundStyle(.white)
        .background(.blue, in:
            RoundedRectangle(cornerRadius: 16))
}
```

Introducing @State

Updating UI

Let’s take a look at the following code:

```swift
struct ContentView: View {
   var counter: Int = 0

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

            Button("Increment Counter") {
                counter += 1
                //Left side of mutating operator isn't mutable: 'self' is immutable
            }
        }

    }
}
```

Introducing @State

Making it Work

Let’s try adding **@State** before our counter property… our error magically disappears!

Now, pressing *Increment Counter* causes our view to update with the new value.

```
struct ContentView: View {
    @State var counter: Int = 0

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

            Button("Increment Counter") {
                counter += 1
            }
        }

    }
}
```

Introducing @State

So What?

@State tells Swift that this property is part of our **app's state,** or the collection of information it needs to decide what should be displayed and how it should behave.

When an **@State** var changes, SwiftUI redraws the View with the new value

‼ Always make **@State** variables **private**. This will prevent obscure issues!

Practice Using @State

Return to Notion

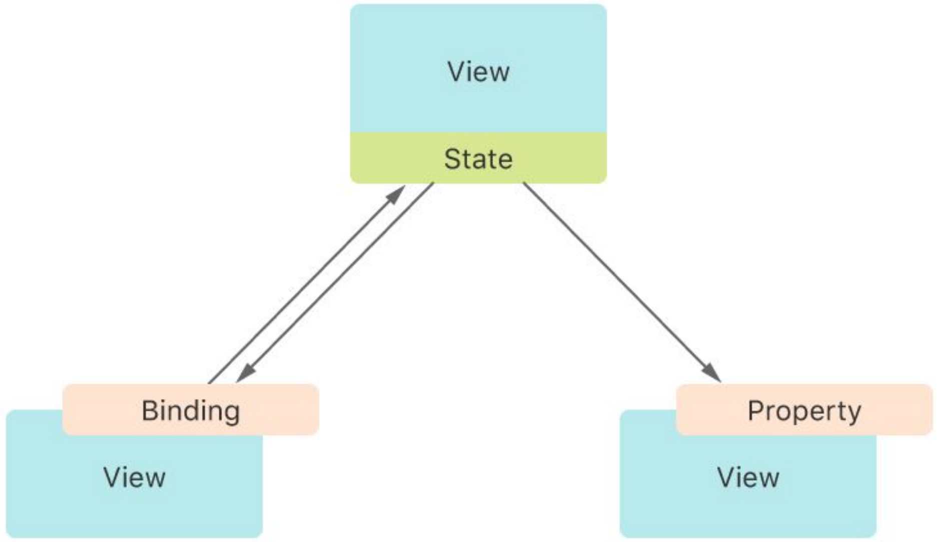
@Binding

@Binding

What is it?

@State variables can be passed to a subview in two ways:

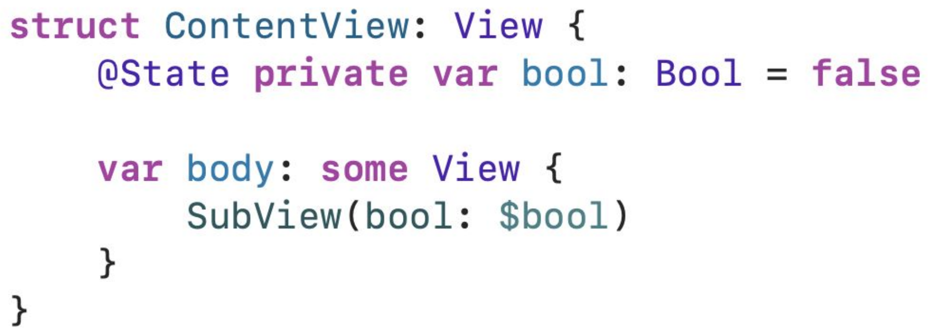
1. As a **property:** read-only
2. As a **binding:** read and write



A View uses **@Binding** when it **does not own** the @State variable it wants to modify.

@Binding
What is it?

When you pass a binding to a subview, you must add a **$** prefix to the @State variable name. This denotes the parameter as a binding with **read-write privileges**.



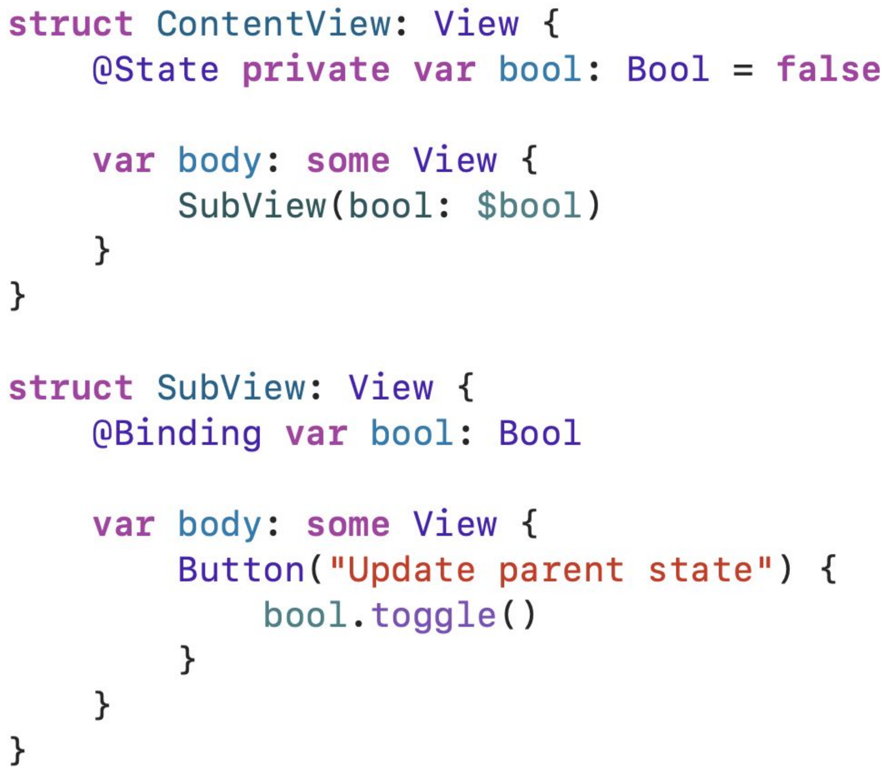
```swift
struct ContentView: View {
    @State private var bool: Bool = false

    var body: some View {
        SubView(bool: $bool)
    }
}
```

Without this **$**, passing the @State var to a subview **only creates a copy** of the value.

@Binding

Example



```swift
struct ContentView: View {
    @State private var bool: Bool = false

    var body: some View {
        SubView(bool: $bool)
    }
}

struct SubView: View {
    @Binding var bool: Bool

    var body: some View {
        Button("Update parent state") {
            bool.toggle()
        }
    }
}
```

@State vs. @Binding

What's the difference?

@State

- The single source of truth for a piece of data.
- Owned and stored by the view itself.
- Changes automatically refresh the view.

Parent owns @State

@Binding

- A reference to a source of truth.
- Can read and update the data, but does not store it.
- Used to let child views interact with the parent's state safely.

Child uses @State via @Binding

More User Input

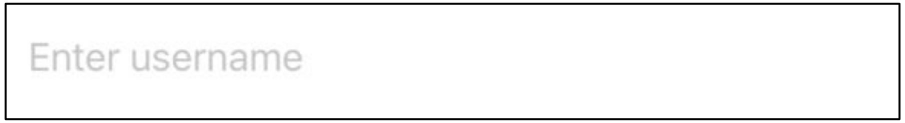
TextField

TextField has two parameters:

1. **Prompt** – A string that appears in the TextField when it's empty.
2. **Text** – A binding to a string that reflects the current text in the field.

```swift
struct ContentView: View {
    @State private var username: String = ""

    var body: some View {
        TextField("Enter username", text: $username)
    }
}
```



Enter username

More User Input

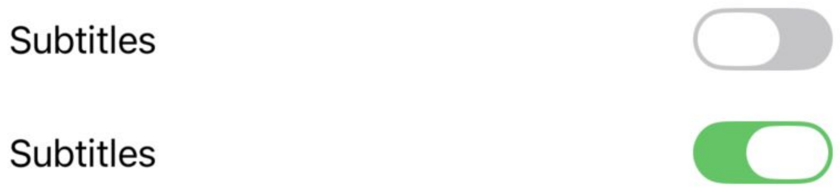
Toggle

Toggle has two parameters:

1. **Label** – A View that appears next to the Toggle describing its purpose
2. **isOn** – A binding to a Bool that reflects the Toggle's current state.

```swift
struct ContentView: View {
    @State private var subtitlesOn: Bool = false

    var body: some View {
        Toggle("Subtitles", isOn: $subtitlesOn)
    }
}
```



Practice Using @Binding

Return to Notion

Countey Demo

Return to Notion