
SwiftUI
Better apps. Less code.

SwiftUI

- Native framework from Apple.
- Declarative view.
- Views are structs.
- SwiftUI uses states.
- SwiftUI creates a binding with the state
- When a states change SwiftUI render a new view.

@State

property wrapper

- SwiftUI manages the storage of any property you declare as a state. When the state value changes, the view invalidates its appearance and recomputes the body. Use the state as the single source of truth for a given view.
- Declares @State as private because you should only access a state property from inside the view's body
- To pass a state property to another view in the view hierarchy, use the variable name with the $prefix operator.

@Binding
property wrapper

- Use a binding to create 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.

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

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```

Observable Object

protocol

- A type of object with a publisher that emits before the object has changed.
- We can add the @ObservedObject
- To get the changes of the object

```
class Contact: ObservableObject {
    @Published var name: String
    @Published var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    func haveBirthday() -> Int {
        age += 1
        return age
    }
}

let john = Contact(name: "John Appleseed", age: 24)
cancellable = john.objectWillChange
    .sink { _ in
        print("\(john.age) will change")
}
print(john.haveBirthday())
// Prints "24 will change"
// Prints "25"
```

@Environment

Property

@Binding

BindableObject

@State