# SwiftUI Interview Prep Sheet — Detailed

## 🟢Beginner-Level

### 1. What is SwiftUI, and how is it different from UIKit?

**Answer:** SwiftUI is Apple's declarative UI framework for iOS, macOS, watchOS, and tvOS. Unlike UIKit (imperative), you declare *what* the UI should be and SwiftUI updates the interface automatically when data changes.

```
struct ContentView: View {
    var body: some View {
        Text("Hello, SwiftUI!")
            .padding()
            .foregroundColor(.blue)
    }
}
```

---

### 2. What is a `View` in SwiftUI?

**Answer:** A `View` is a struct conforming to the `View` protocol. The `body` property describes the UI hierarchy.

```
struct MyView: View {
    var body: some View {
        VStack {
            Text("Hello")
            Text("World")
        }
    }
}
```

---

### 3. What is `.ignoresSafeArea()`?

**Answer:** It allows a view to extend into system safe areas such as the notch or home indicator.

```
Color.red
    .ignoresSafeArea(.all)
```

---

### 4. What is the difference between `.frame()` and `.padding()`?

**Answer:** - `.frame()` sets the view's size explicitly. - `.padding()` adds spacing *inside* or *around* the view without changing intrinsic content.

```
Text("Hello")
    .frame(width: 100, height: 50)
    .padding(10)
```

---

### 5. What is a `Spacer()` and how does it work?

**Answer:** `Spacer()` expands to fill available space within a container like `HStack` or `VStack`.

```
HStack {
    Text("Left")
    Spacer()
    Text("Right")
}
```

---

## 🟡Intermediate-Level

### 6. What is `@State` and when should you use it?

**Answer:** `@State` is a property wrapper for storing view-local state. SwiftUI automatically updates the UI when the value changes.

```
struct CounterView: View {
    @State private var count = 0

    var body: some View {
        VStack {
            Text("Count: \(count)")
            Button("Increment") { count += 1 }
        }
    }
}
```

---

### 7. Difference between `@Binding` and `@State`?

**Answer:** - `@State` stores local mutable data inside a view. - `@Binding` provides a two-way reference to a `@State` variable from a parent view.

```swift
struct ParentView: View {
    @State private var isOn = false
    var body: some View {
        ToggleView(isOn: $isOn)
    }
}

struct ToggleView: View {
    @Binding var isOn: Bool
    var body: some View {
        Toggle("Switch", isOn: $isOn)
    }
}
```

---

### 8. Difference between `@ObservedObject` and `@StateObject`?

**Answer:** - `@ObservedObject` subscribes to an ObservableObject but **does not own** it; recreating the view can recreate the object. - `@StateObject` **owns** the ObservableObject and persists it across view reloads.

```swift
class CounterVM: ObservableObject {
    @Published var count = 0
}

struct ContentView: View {
    @StateObject var vm = CounterVM() // Owned and persisted
}
```

---

### 9. What is `@EnvironmentObject`?

**Answer:** `@EnvironmentObject` allows sharing an ObservableObject across many views without passing it manually.

```swift
class Settings: ObservableObject { @Published var isDark = false }

struct Parent: View {
    @StateObject var settings = Settings()
    var body: some View {
        Child().environmentObject(settings)
    }
}

struct Child: View {
```

```
    @EnvironmentObject var settings: Settings
}
```

---

### 10. How do you navigate between views?

**Answer:** Use `NavigationStack` (iOS 16+) or `NavigationView` + `NavigationLink`.

```
NavigationStack {
    VStack {
        NavigationLink("Go to Detail", destination: DetailView())
    }
}
```

---

### 11. How do you create a tab bar?

**Answer:** Use `TabView` with `.tabItem()`.

```
TabView {
    Text("Home").tabItem { Label("Home", systemImage: "house") }
    Text("Profile").tabItem { Label("Profile", systemImage: "person") }
}
```

---

## 🔵Advanced-Level

### 12. What is `ViewBuilder`?

**Answer:** `@ViewBuilder` allows building multiple child views in a closure without returning a single explicit view.

```
struct MyStack<Content: View>: View {
    let content: () -> Content
    init(@ViewBuilder content: @escaping () -> Content) { self.content =
content }
    var body: some View { VStack { content() } }
}
```

---

### 13. How do you create custom view modifiers?

**Answer:** Conform to `ViewModifier` and implement `func body(content: Content) -> some View`.

```swift
struct RedBorder: ViewModifier {
    func body(content: Content) -> some View {
        content.padding().border(Color.red)
    }
}

Text("Hello").modifier(RedBorder())
```

---

### 14. How do you create reusable custom views?

**Answer:** Create a struct conforming to `View` with parameters.

```swift
struct CustomButton: View {
    var title: String
    var action: () -> Void
    var body: some View {
        Button(title, action: action)
            .padding().background(Color.blue).foregroundColor(.white)
            .cornerRadius(8)
    }
}
```

---

### 15. What is the difference between `ZStack`, `VStack`, and `HStack`?

**Answer:** - `VStack`: vertical layout - `HStack`: horizontal layout - `ZStack`: layered (stacked) layout

```swift
ZStack {
    Color.gray
    Text("On top")
}
```

---

### 16. How do you apply conditional modifiers?

**Answer:** Use `if` inside modifier chain or ternary operators.

```swift
Text("Hello")
    .foregroundColor(isRed ? .red : .blue)
```

---

### 17. How do you animate a view?

**Answer:** Use `.animation()` or `.withAnimation {}`.

```
@State var scale: CGFloat = 1.0

Button("Animate") {
    withAnimation {
        scale += 0.5
    }
}
.scaleEffect(scale)
```

---

### 18. How do you create a custom navigation bar?

**Answer:** Use `.toolbar` modifier and `.navigationBarTitleDisplayMode()`.

```
NavigationStack {
    Text("Content")
        .toolbar {
            ToolbarItem(placement: .navigationBarTrailing) {
                Button("Edit") {}
            }
        }
        .navigationTitle("Dashboard")
        .navigationBarTitleDisplayMode(.inline)
}
```

---

### 19. How do you handle different screen sizes?

**Answer:** Use `GeometryReader`, `frame(maxWidth: .infinity)`, `LayoutPriority`, and adaptive spacers.

```
GeometryReader { geo in
    VStack {
        Text("Width: \(geo.size.width)")
    }
}
```

---

### 20. How do you integrate UIKit views?

**Answer:** Use `UIViewRepresentable` for views and `UIViewControllerRepresentable` for controllers.

```
struct MyUIKitLabel: UIViewRepresentable {
    func makeUIView(context: Context) -> UILabel { UILabel() }
    func updateUIView(_ uiView: UILabel, context: Context) { uiView.text =
"Hello UIKit" }
}
```

---

### 21. How do you create a scrollable horizontal/vertical list?

**Answer:** Use `ScrollView` combined with `LazyVStack` or `LazyHStack`.

```
ScrollView(.vertical) {
    LazyVStack {
        ForEach(0..<100) { i in
            Text("Item \(i)")
        }
    }
}
```

---

### 22. How do you implement a page/tab-like horizontal scroll?

**Answer:** Use `TabView` with `.tabViewStyle(.page)`.

```
TabView {
    ForEach(0..<5) { i in
        Text("Page \(i)")
    }
}
.tabViewStyle(.page)
```

---

### 23. How do you handle view lifecycle events?

**Answer:** Use `.onAppear` and `.onDisappear` modifiers.

```
Text("Hello")
    .onAppear { print("Appeared") }
    .onDisappear { print("Disappeared") }
```

---

### 24. How do you customize a `List` row?

**Answer:** Use `listRowBackground` and `listRowInsets` modifiers.

```
List(0..<10) { i in
    Text("Row \(i)")
        .listRowBackground(Color.yellow.opacity(0.3))
}
```

---

### 25. How do you create a gradient background?

**Answer:** Use `LinearGradient` or `RadialGradient`.

```
LinearGradient(colors: [.red, .blue], startPoint: .top, endPoint: .bottom)
    .ignoresSafeArea()
```

---

### 26. How do you clip a view into a shape?

**Answer:** Use `.clipShape()` modifier.

```
Image(systemName: "star.fill")
    .resizable()
    .frame(width: 50, height: 50)
    .clipShape(Circle())
```

---

### 27. How do you overlay one view on top of another?

**Answer:** Use `.overlay()` modifier.

```
Rectangle()
    .fill(Color.blue)
    .frame(width: 100, height: 100)
    .overlay(Text("Top"))
```

---

### 28. How do you use `GeometryReader`?

**Answer:** `GeometryReader` provides the size and position of its child view.

```
GeometryReader { geo in
    Text("Width: \(geo.size.width)")
}
```

---

### 29. How do you implement conditional views?

**Answer:** Use `if-else` inside the body or a `Group`.

```
VStack {
    if isLoggedIn {
        Text("Welcome")
    } else {
        Text("Please Login")
    }
}
```

---

### 30. How do you handle safe area insets?

**Answer:** Use `.padding(.top, safeAreaInsets.top)` or `.ignoresSafeArea()`.

```
Color.green.ignoresSafeArea(edges: .all)
```

---

*(The document can continue further for advanced layout, animations, and performance optimizations as needed.)