📘 Unit 1 – Basic SwiftUI Views and Controls

Well-Structured, Fully Formatted Reading Material

✂️ SwiftUI vs UIKit

SwiftUI

- Declarative
- Less code, clean syntax
- Reactive state updates (`@State`, `@Binding`)
- Cross-platform (iOS, macOS, watchOS, tvOS)

UIKit

- Imperative
- Verbose code
- iOS-only
- Manual UI updates

Advantages of SwiftUI

- Declarative syntax → UI auto-updates with state
- Less code, higher readability
- Reactive architecture
- Works across Apple platforms

Code Example

```
@State private var isOn = false
Toggle("Enable Feature", isOn: $isOn)
Text(isOn ? "Enabled" : "Disabled")
```

✒️ ZStack – Overlay Views

Concept

- ZStack overlays views on top of each other.
- Useful for layered UI (backgrounds, badges, overlays).

Code Example

```
ZStack {
    Image("background")
        .resizable()
        .scaledToFill()
    Text("Hello Beast")
        .font(.title)
        .foregroundColor(.white)
}
```

🛏️Label and Image View

Label View

- Combines icon + text.

Image View

- Displays assets or SF Symbols.
- Styling using modifiers like `.clipShape()`, `.frame()`, `.font()`.

Code Example

```
Label("Profile", systemImage: "person.circle")
    .font(.title)
    .foregroundColor(.blue)

Image("banner")
    .resizable()
    .scaledToFit()
    .frame(height: 120)
    .clipShape(RoundedRectangle(cornerRadius: 12))
```

🌂Toggle and Stepper

Toggle

- A switch used for Boolean values.

Stepper

- Increments/decrements numeric values.

Code Example

```
@State private var isOn = true
@State private var quantity = 1

VStack {
    Toggle("Notifications", isOn: $isOn)
    Stepper("Quantity: \(quantity)", value: $quantity, in: 1...10)
}
```

🌡️ Swift Prevents Common Programming Errors

- **Optionals** → prevent null pointer issues
- **Type safety** → strong typing avoids incorrect values
- **ARC (Automatic Memory Management)** → prevents leaks
- **Error handling** → `try/catch` ensures safe execution
- **Immutability** (`let`) → avoids accidental changes

🎒Image with Multiple Modifiers + ViewModifier

Concept

- Apply modifiers:
  `.resizable()`, `.scaledToFill()`, `.frame()`, `.clipShape()`, `.opacity()`.

Code Example

```
struct StyledImage: View {
    var body: some View {
        Image("profile")
            .resizable()
            .scaledToFill()
            .frame(width: 120, height: 120)
            .clipShape(Circle())
            .opacity(0.9)
            .modifier(ShadowStyle())
    }
}
```

```swift
struct ShadowStyle: ViewModifier {
    func body(content: Content) -> some View {
        content.shadow(radius: 5)
    }
}
```

🎓ViewModifier (Group Styles)

Purpose

- Reuse multiple styling modifiers.

Example

```swift
struct TitleStyle: ViewModifier {
    func body(content: Content) -> some View {
        content
            .font(.title)
            .foregroundColor(.purple)
    }
}

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

🎩Optional Type in Swift

Concept

- Optional can store a value or `nil`.
- Helps prevent runtime crashes.

Example

```swift
var name: String? = "Beast"
if let safeName = name {
    print("Hello \(safeName)")
} else {
    print("No name")
}
```

🎽 Custom Parameter Attribute – @ViewBuilder

Purpose

- Allows closure to return multiple views.
- Used in stacks and custom views.

Code Example

```
struct CardView<Content: View>: View {
    let content: Content
    init(@ViewBuilder content: () -> Content) {
        self.content = content()
    }
    var body: some View {
        VStack { content }
            .padding()
            .background(Color.gray.opacity(0.2))
            .cornerRadius(8)
    }
}

CardView {
    Text("Title")
    Text("Subtitle")
}
```

🕑Text View – lineLimit & truncationMode

Concept

- Displays read-only text.
- Control overflow using `.lineLimit()` and `.truncationMode()`.

Example

```
Text("This is a very long sentence that may not fit.")
    .lineLimit(1)
    .truncationMode(.tail)
```

✂️✂️Arrays in Swift

Two Syntaxes

```
var numbers: [Int] = [1, 2, 3]
var numbers = Array<Int>([4, 5, 6])
```

Example

```
var integers: [Int] = [10, 20, 30]
for num in integers {
    print(num)
}
```

✂️✒️TextField and TextEditor

Concept

- **TextField** → single-line text input.
- **TextEditor** → multi-line input.

Example

```
@State private var name = ""
@State private var notes = ""

VStack {
    TextField("Enter name", text: $name)
        .textFieldStyle(.roundedBorder)
    TextEditor(text: $notes)
        .frame(height: 100)
        .border(Color.gray)
}
```

✂️🛏️Adding SwiftUI to UIKit

Concept

Use `UIHostingController` to embed SwiftUI inside UIKit.

Code Example

```
import SwiftUI

struct SwiftUIView: View {
    var body: some View {
        Text("Hello from SwiftUI")
    }
}

let controller = UIHostingController(rootView: SwiftUIView())
present(controller, animated: true)
```

🔗 Unit-1 Summary

- SwiftUI vs UIKit → declarative, reactive, cross-platform.
- ZStack → overlays views.
- Label & Image → icon + text.
- Toggle & Stepper → boolean + numeric controls.
- Swift safety → optionals, ARC, type safety.
- Image modifiers + ViewModifier → reusable styles.
- Optional type → safe `nil` handling.
- @ViewBuilder → supports multiple views.
- Text view → lineLimit, truncation.
- Arrays → two syntax styles.
- TextField & TextEditor → input controls.
- UIKit integration → via UIHostingController.

👙 End of Unit 1