MCA Sem-3
Mobile Application Development with iOS

Introduction

Course Overview

- 1. Basic SwiftUI Views and Controls
  - iOS Applications
  - Swift Programming Basics
  - Laying Out Components
  - Text, Image, Button, Segmented Control, TextField, Picker
  - View Modifier, View Builder, Symbol
  - Integrating UIKit in to SwiftUI, Adding SwiftUI to Legacy UIKit App

Course Overview

- 2. Lists Views, Scroll Views, and Advanced Components
  - Scroll Views, Creating List of Static Items
  - Custom Rows in a List, Adding/Deleting Rows
  - Editable List View, Editable Collection, Searchable List
  - LazyHStack and LazyVStack, LazyHGrid and LazyVGrid
  - Scrolling Programmatically, Expanding Lists
  - Disclosure Groups, Create SwiftUI Widgets

Dharmendra Bhatti 3

Course Overview

- 3. Navigation
  - Using NavigationStack
  - Typed Data-driven Navigation with NavigationStack
  - Untyped Data-driven Navigation with NavigationStack
  - Working with NavigationSplitView
  - Using TabView
  - Programmetically switching tabs on a TabView

Dharmendra Bhatti 4

Course Overview

- 4. SwiftUI with Data
  - Using @State to drive a View's behavior
  - Using @Binding to pass a state variable to child views
  - Core location wrapper @ObservedObject
  - Preserve model's life cycle using @StateObject
  - Sharing state object using @EnvironmentObject
  - Using Observation to manage model data

Dharmendra Bhatti 5

Course Overview

- 5. Combine and Firebase
  - Introduction to Combine
  - Managing Memory and Validating a Form
  - Fetching Remote Data and Visualizing it
  - Sign in with Apple
  - Integrating Firebase into SwiftUI
  - Google Sign in

Dharmendra Bhatti 6

Course Overview

- 6. Core Data and Swift Data
  - Integrating Core Data with SwiftUI
  - Showing Core Data Objects with @FetchRequest
  - Adding Core Data Objects from a SwiftUI View
  - Filtering Core Data Objects with Predicate
  - Deleting Core Data Objects, @SectionedFetchRequest
  - Working with SwiftData

Dharmendra Bhatti 7

Mobile App Development

- Design and develop software for mobile devices like a phone or tablet.
- Understand how mobile apps are different from conventional desktop apps.
- Learn how to use the language, OS, IDE and frameworks to effectively create mobile applications.
- Realize the full potential of your app by utilizing the capabilities of mobile device.

Dharmendra Bhatti 8

What makes mobile development different?

- Desktop = large screen, mouse, stationary
- Laptop = smaller desktop UX
- Mobile =
  - Hand-held, small, hi res, screen
  - Multi-touch: Gesture
  - Anywhere: not just sitting
  - Aware: sensors
  - Always on and connected

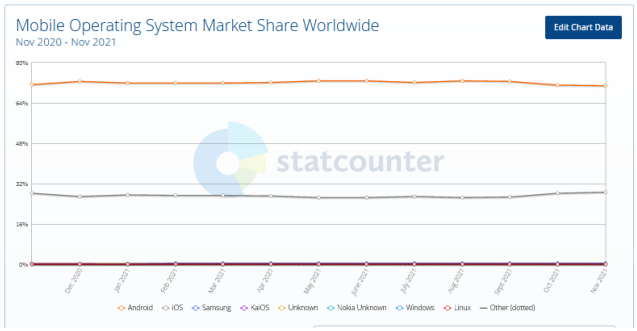
Dharmendra Bhatti 9

Special Development Considerations

- Limited operating memory
- Small screen
- Real-time application constraints
- Application response time
- Suspend/Resume
- System and application reliability

Dharmendra Bhatti 10

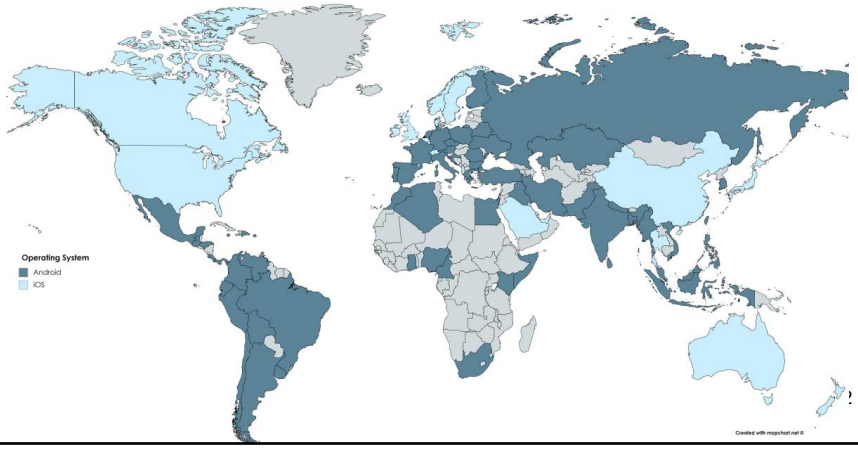
Mobile OS Market Share Nov20-Nov21



- https://gs.statcounter.com/os-market-share/mobile/worldwide

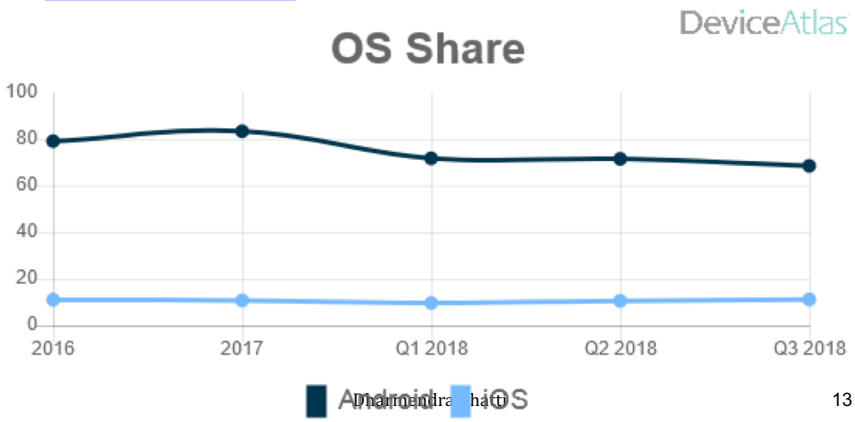
Android v iOS market share

- https://deviceatlas.com/blog/android-v-ios-market-share



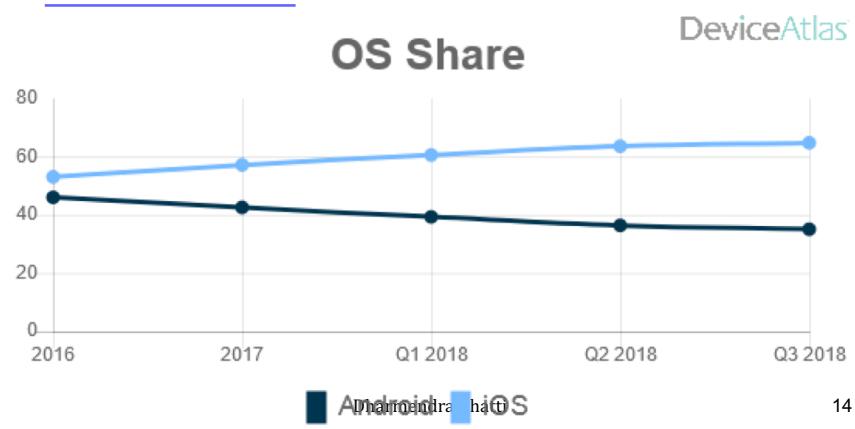
India – market share 2018

- https://deviceatlas.com/blog/android-v-ios-market-share



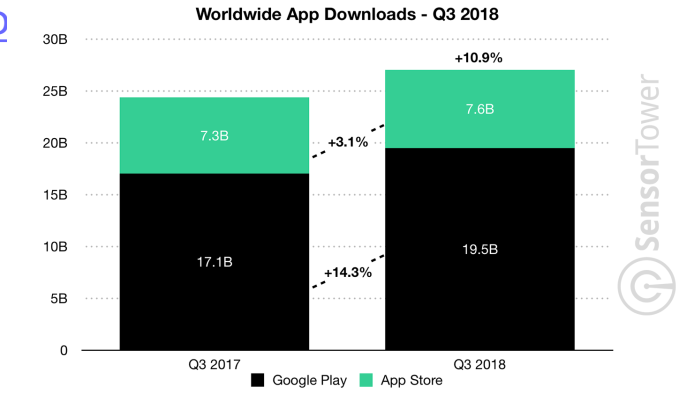
USA – market share 2018

- https://deviceatlas.com/blog/android-v-ios-market-share



Android v iOS App Downloads 2018

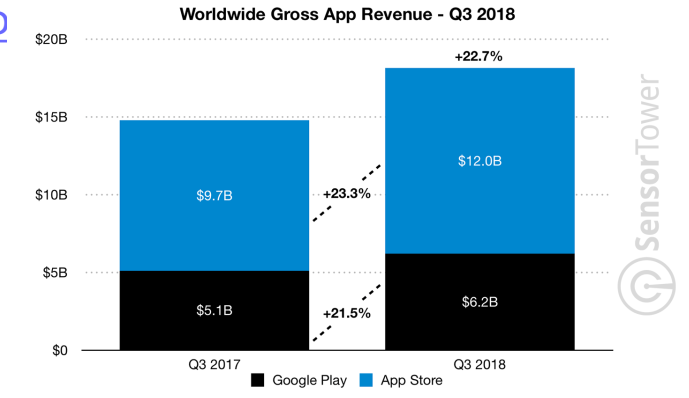
- https://techcrunch.com/2018/10/11/app-store-generated-93-more-revenue-than-go



15

Android v iOS revenue 2018

- https://techcrunch.com/2018/10/11/app-store-generated-93-more-revenue-than-go



16

iOS

- What's in iOS
  - Core OS
    - OSX Kernel, Mach 3, BSD, Sockets, Security, Power Management, Keychain Access, Certificates, File System, Bonjour
  - Core Services
    - Collections, Address Book, Networking, File Access, SQLite, Core Location, Net Services, Threading, Preferences, URL Utilities
  - Media
    - Core Audio, OpenAL, Audio Mixing, Audio Recording, Video Playback, Images, PDF, Quartz, Core Animation, OpenGL ES
  - Cocoa Touch
    - Multi-Touch, Core Motion, View Hierarchy, Localization, Controls, Alerts, WebView, Map Kit, Image Picker, Camera

Dharmendra Bhatti 17

SWIFT PROGRAMMING

Dharmendra Bhatti 18

Swift Overview

- Swift was introduced in 2014 as the "new way to create apps for iPhone and iPad"
  - Objective-C without the C
  - Descendant to Objective-C
  - No pointers
  - Functions as first-class objects
  - Influenced from many modern languages:
    - JavaScript, Objective-C, C#, Python, Scala, etc…

Dharmendra Bhatti 19

Swift defies away large classes of common programming errors

- Variables are **always initialized** before use.
- Array indices are checked for **out-of-bounds** errors.
- Integers are checked for **overflow**.
- **Optionals** ensure that nil values are handled explicitly.
- Memory is managed automatically.
- **Error handling** allows controlled recovery from unexpected failures.

Dharmendra Bhatti 20

Types in Swift

- Swift types can be arranged into three basic groups:
  - *structures*,
  - *classes*, and
  - *enumerations*

Structures

```
struct MyStruct {
    // properties
    // initializers
    // methods
}
```

Enumerations

```
enum MyEnum {
    // properties
    // initializers
    // methods
}
```

Classes

```
class MyClass: SuperClass {
    // properties
    // initializers
    // methods
}
```

Dharmendra Bhatti 21

Types in Swift

- Primitive Swift types are structures:

Numbers:	**`Int`**, **`Float`**, **`Double`**
Boolean:	**`Bool`**
Text:	**`String`**, **`Character`**
Collections:	**`Array<Element>`**, **`Dictionary<Key:Hashable,Value>`**, **`Set<Element:Hashable>`**

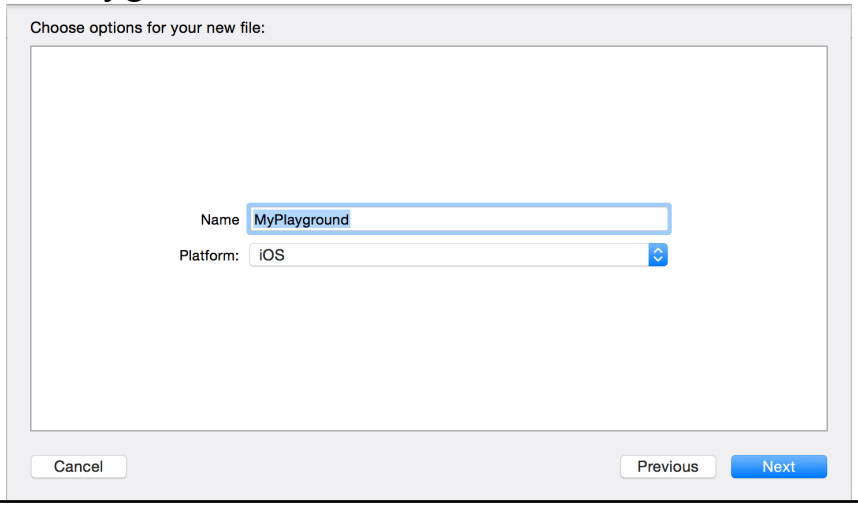
Dharmendra Bhatti 22

Using the Swift Playground

- Apple has provided a Playground for experimenting with Swift features inside Xcode
  - Write code and receive immediate result
  - Great for exploring Swift features and research
  - Allows fast testing of features and syntax

Configuring a playground

- In Xcode, select File → New → Playground



Swift Syntax Overview

- Swift syntax:
  - Variables and constants
  - Control structures
    - If-else, loops, switch
  - Data structures
    - Arrays, dictionaries, sets
  - Classes and structures
    - Init functions, methods, properties, fields
  - OOP
    - Protocols, extensions, inheritance

Dharmendra Bhatti 25

Data types, Variables and Constants

var and let, primitive and reference, functions

Dharmendra Bhatti 26

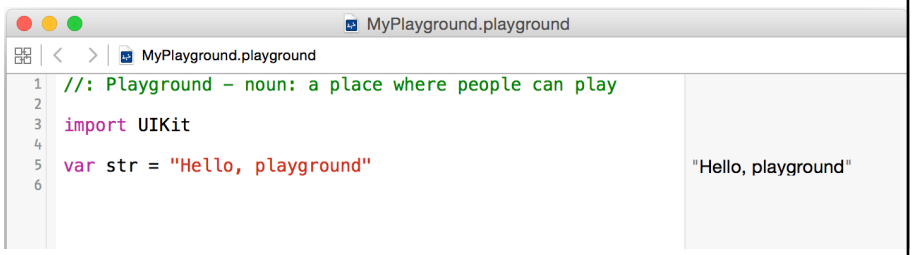
Data Types in Swift

- Swift supports the common data types, coming from Objective-C
  - `Int` for creating integers numbers
  - `UInt` for creating unsigned integer numbers
  - `Float`, `Double` for floating-point numbers
  - `Bool` for Boolean values
  - `[]` for array, `[Int]` for array of integers
- Swift supports both structures and classes
  - Structures create primitive types
  - Classes create reference types

Data Types, Variables and Constants

- Swift provides two keywords for creating variables:
  - `let` and `var`
- `var` creates a mutable variable
  - Its value can be changed any time
  - The type can be provided or inferred
- `let` creates immutable variable
  - If the type is struct, its value cannot be changed
  - If the type is object, it can be changed only internally

Variable



- the value of str can be changed from its initial value

```
var str = "Hello, playground"                "Hello, playground"
str = "Hello, Swift"                         "Hello, Swift"
```

Dharmendra Bhatti 29

var, let

- Can't change value of constant

```
var str = "Hello, playground"                "Hello, playground"
str = "Hello, Swift"                         "Hello, Swift"
let constStr = str                           "Hello, Swift"
constStr = "Hello, world"
```

Dharmendra Bhatti 30

Inferring types

- **Option-Click** to infer type using Xcode's Quick Help.

```
var str = "Hello, playground"
str = "Hello, Swift"
let constStr = str
```

```
Declaration   let constStr: String
Declared In   MyPlayground.playground
```

- If constant or variable has an initial value, you can rely on type inference.

Dharmendra Bhatti 31

Specifying types

- var nextYear: Int
- var bodyTemp: Float
- var hasPet: Bool

Dharmendra Bhatti 32

Number and Boolean types

- Int
- Float
- Double
- Float80
- Bool

Collection types

- The Swift standard library offers three collections:
  - *arrays*
  - *dictionaries*
  - *sets*

- **var arrayOfInts: Array<Int>**
  or
- **var arrayOfInts: [Int]**

Collection types

- A **dictionary** is an unordered collection of key-value pairs.
- var dictionaryOfCapitalsByCountry: Dictionary<String,String>
- OR
- var dictionaryOfCapitalsByCountry: [String:String]

Dharmendra Bhatti 35

Collection types

- A set is similar to an array in that it contains a number of elements of a certain type.
- var winningLotteryNumbers: Set<Int>

Dharmendra Bhatti 36

Initializers

- Initializers are responsible for preparing the contents of a new instance of a type.

```
let emptyString = String()                          ""
let emptyArrayOfInts = [Int]()                      0 elements
let emptySetOfFloats = Set<Float>()                 0 elements
```

Other types have default values:

```
let defaultNumber = Int()                           0
let defaultBool = Bool()                            false
```

Dharmendra Bhatti 37

Initializers

```
let number = 42                                     42
let meaningOfLife = String(number)                  "42"
```

To create a set, you can use the **Set** initializer that accepts an array literal:

```
let availableRooms = Set([205, 411, 412])           {412, 205, 411}
```

```
let defaultFloat = Float()                          0.0
let floatFromLiteral = Float(3.14)                  3.14
```

Dharmendra Bhatti 38

Properties

- A *property* is a value associated with an instance of a type.

```
let countingUp = ["one", "two"]                    ["one", "two"]
let secondElement = countingUp[1]                  "two"
countingUp.count                                   2
...
let emptyString = String()
emptyString.isEmpty                                true
```

Dharmendra Bhatti 39

Instance methods

- An *instance method* is a function that is specific to a particular type and can be called on an instance of that type.

```
var countingUp = ["one", "two"]                    ["one", "two"]
let secondElement = countingUp[1]                  "two"
countingUp.count
countingUp.append("three")                         ["one", "two", "three"]
```

Dharmendra Bhatti 40

Operators

Supports all standard Arithmetic, Comparison, Logical, Bitwise, Assignment, Range (0..<3) Operators

Supports Ternary Conditional (`Condition ? X : Y`). Unary Plus and Minus

41

Strings

String Interpolation

```
let multiplier = 3
let message = "\(multiplier) times 2.5 is \(Double(multiplier) * 2.5)"
// message is “3 times 2.5 is 7.5”
```

Can be applied to print statements

String Concatenation

```
let str = "Hello" + " swift lovers" // "Hello swift lovers"
```

Methods on string

```
print(str.characters.count) // 18
```

42

Optionals?

- Swift types can be *optional*, which is indicated by appending ? to a type name.

- var anOptionalFloat: Float?
- var anOptionalArrayOfStrings: [String]?
- var anOptionalArrayOfOptionalStrings: [String?]?

- An optional lets you express the possibility that a variable may not store a value at all.

- The value of an optional will either be an instance of the specified type or nil.

43

Optionals?

Normal Float

- var reading1: Float
- var reading2: Float
- var reading3: Float

Optional Float

- var reading1: Float?
- var reading2: Float?
- var reading3: Float?

- cannot use these optional floats like non-optional floats
-
- reading1 = 9.8
- reading2 = 9.2
- reading3 = 9.7
- **let avgReading = (reading1 + reading2 + reading3) / 3**

Dharmendra Bhatti 44

Optionals?

- optionals require **unwrapping**
- **let avgReading = (reading1! + reading2! + reading3!) / 3**
- OR

```
if let r1 = reading1,
    let r2 = reading2,
    let r3 = reading3 {
        let avgReading = (r1 + r2 + r3) / 3
} else {
    let errorString = "Instrument reported a reading that was nil."
}
```

45

Subscripting dictionaries

- The result of subscripting a dictionary is an optional:

```
let nameByParkingSpace = [13: "Alice", 27: "Bob"]          [13: "Alice", 27: "Bob"]
let space13Assignee: String? = nameByParkingSpace[13]      "Alice"
let space42Assignee: String? = nameByParkingSpace[42]      nil
```

- OR

```
if let space13Assignee = nameByParkingSpace[13] {
    print("Key 13 is assigned in the dictionary!")
}
```

46

Defining Functions in Swift

- Functions in Swift have the following syntax:
  - Function returning void:

```
func printMessage(message: String) {
  println(String(format: "Message: %@", message))
}
```

- **Function returning integer:**

```
func sum(x: Int, y: Int) -> Int {
  return x + y
}
```

- **Function with parameters put inside an array:**

```
func sum(numbers: Int…) -> Int {
  var sum = 0
  for number in numbers { sum += number}
  return sum
}
```

Dharmendra Bhatti 47

Functions - I

```
func sayHello(personName: String) ->
  String {
     return "Hello, " + personName +
  "!"
}
print(sayHello("Bob"))
// Prints "Hello, Bob!
```

Supports nested functions → Create function inside a function!

48

Functions - II

Functions with Multiple Return Values

```
func minMax(array: [Int]) -> (min: Int, max: Int)
  {
    var currentMin = array[0]
    var currentMax = array[0]
    // some logic ...
    return (currentMin, currentMax)
}

let bounds = minMax([8, -6, 2, 109, 3, 71])
print("min is \(bounds.min) and max is
  \(bounds.max)")
```

49

Control Flows - unwrapping optionals

```
let possibleNumber = "123" // possibleNumber is type
  String

let convertedNumber = Int(possibleNumber) //
  convertedNumber is type Int?

if convertedNumber != nil {
  // now that we know that convertedNumber has a value,
  force unwrap using "!"
     print("convertedNumber has integer value of
  \(convertedNumber!).")
}
```

50

Control Flows - Optional Binding - I

Use to find out whether an optional contains a value, and if so, to make that value available as a temporary constant or variable.

Can be used with if and while statements

```
if let actualNumber = Int(possibleNumber) {
      print("\"\(possibleNumber)\" has an integer value of
   \(actualNumber)")
} else {
      print("\"\(possibleNumber)\" could not be converted
   to an integer")
}
```

51

Control Flows - Optional Binding - II

Can also include multiple optional bindings in a single `if` statement

```
if let firstNumber = Int("4"), secondNumber =
   Int("42") where firstNumber < secondNumber
{
    print("\(firstNumber) < \(secondNumber)")
}
// Prints "4 < 42"
```

52

Control Flow Structures: if-else and switch

- Swift supports the standard control flow structures:

if-else

```
if condition {
  //run code
}
else if condition2 {
  //run other code
}
else {
  //run third code
}
```

switch

```
switch language {
case "Objective-C", "Swift":
  println("iOS")
case "Java":
  println("Android")
case "C#":
  println("Windows Phone")
default:
  println("Another platform")
}
```

Dharmendra Bhatti 53

Control Flows - Loops

Swift supports `for`, `for-in`, `while`, and `do...while` loops

```
for var index = 0; index < 3; index += 1 { … }

for item in someArray { … }

while index < 20 { … }

do { … } while index < 20

for i in 0..<3 { /* this will loop 3 times */ }
```

54

for loops

```
let range = 0..<countingUp.count
for i in range {
    let string = countingUp[i]
    // Use 'string'
}

for string in countingUp {
    // Use 'string'
}

for (i, string) in countingUp.enumerated() {
    // (0, "one"), (1, "two")
}
```

55

Enumerations

- An enumeration is a type with a discrete set of values.

```
enum PieType {
    case apple
    case cherry
    case pecan
}

let favoritePie = PieType.apple
```

56

Classes and Structures

Introducing the OOP principles in Swift

Swift Classes

- Swift is an OOP language
  - Has classes, structures and protocols

```
class Square{
  var x: Float
  var y: Float
  var side: Float
  init(x: Float, y: Float, side: Float){
    self.x = x
    self.y = y
    self.side = side
  }

  func calcArea() -> Float{
    return self.side * self.side
  }
}
```

Protocols in Swift

- Protocols in Swift are pretty much the same as in Objective-C
  - Almost like interfaces in C# and Java
  - They provide a public interface for the conforming classes to implement

```
protocol Movable{
  func moveToX(x: Int, y: Int)
  func moveByDx(dx:Int, dy: Int)
}
class Shape: Movable{
  func moveToX(x: Int, y: Int){
    //implementation
  }
  func moveByDx(dx: Int, dy: Int){
    //implementation
  }
}
```

Dharmendra Bhatti 59

Why SwiftUI?

- SwiftUI apps can work alongside UIKit apps
- Industry adoption
- Low learning curve
- Live previews increase speed
- Declarative programming

Dharmendra Bhatti 60

SwiftUI Layout

- **VStack** - Vertical stacks that arrange views above and below another view
- **HStack** - Horizontal stacks that arrange views side by side
- **ZStack** - A stack that overlays views directly over each other

SwiftUI Layout

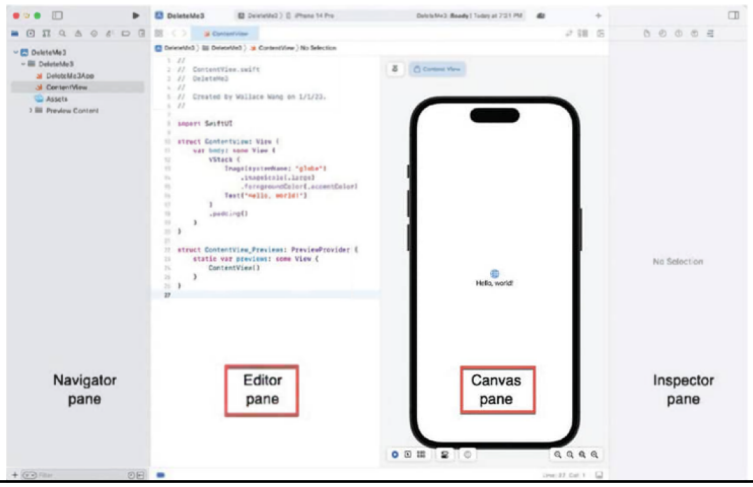
- VStack, HStack, ZStack

SwiftUI Layout

- A stack can only contain a maximum of ten (10) views
- you can embed stacks inside of stacks to display as many views as necessary
-

TheStacks

- Create new SwiftUI project “TheStacks”



TheStacks

- When creating a user interface in SwiftUI, you have three options:
  - Type Swift code in the Editor pane.
  - Drag and drop a view (such as a button) into your Swift code in the Editor pane.
  - Drag and drop a view (such as a button) into the Canvas pane.

Dharmendra Bhatti 65

TheStacks

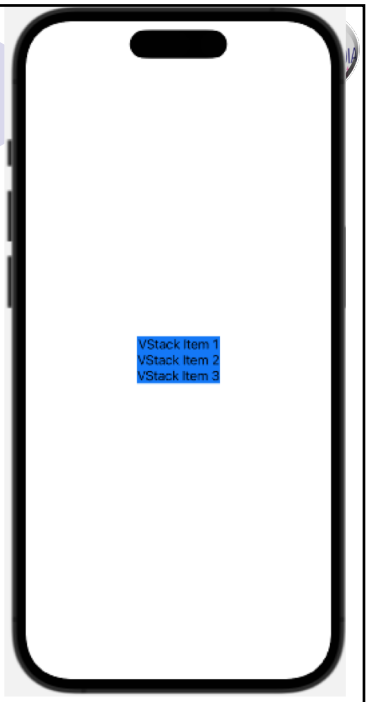
- Replace body content

```
8  import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         VStack {
13             Text("VStack Item 1")
14             Text("VStack Item 2")
15             Text("VStack Item 3")
16         }
17         .background(.blue)
18     }
19 }
```

66

TheStacks

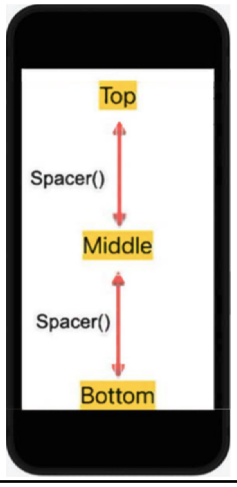
- VStack with three items



Dharmendra Bhatti

TheStacks

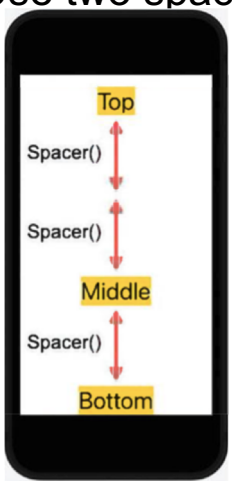
- A spacer acts like a spring that pushes two views as far apart as possible.
- Syntax to use a spacer:
- Spacer()



Dharmendra Bhatti

TheStacks

- Use two spacers to push view further



Dharmendra Bhatti 69

TheStacks

- Add a Spacer and a Divider between VStack Item 2 and VStack Item 3

```
10 struct ContentView: View {
11     var body: some View {
12         VStack {
13             Text("VStack Item 1")
14             Text("VStack Item 2")
15             Spacer()
16             Divider()
17                 .background(.black)
18             Text("VStack Item 3")
19         }
20         .background(.blue)
21     }
22 }
```

70

TheStacks

- Vstack
- +
- Spacer
- +
- Divider

71

TheStacks

- The padding modifier adds space around a user interface view.
- By default, the padding modifier adds space around the top, bottom, leading (left), and trailing (right) sides of a view.
- To use the padding modifier, just add the following after any view:
- . padding()

Dharmendra Bhatti 72

TheStacks

- VStack (spacing: 10)
- VStack (alignment: .leading)
- VStack (alignment: .trailing, spacing: 20)

TheStacks

- Add an HStack
and a Zstack
below VStack Item 3

```
Text("VStack Item 3")
HStack{
    Text("HStack Item 1")
    Divider()
        .background(.black)
    Text("HStack Item 2")
    Divider()
        .background(.black)
    Spacer()
    Text("HStack Item 3")
}
.background(Color.red)
ZStack{
    Text("ZStack Item 1")
        .padding()
        .background(.green)
        .opacity(0.8)
    Text("ZStack Item 2")
        .padding()
        .background(.green)
        .offset(x: 80, y: -400)
}
```

TheStacks

- VStack,
- HStack,
- and ZStack

75

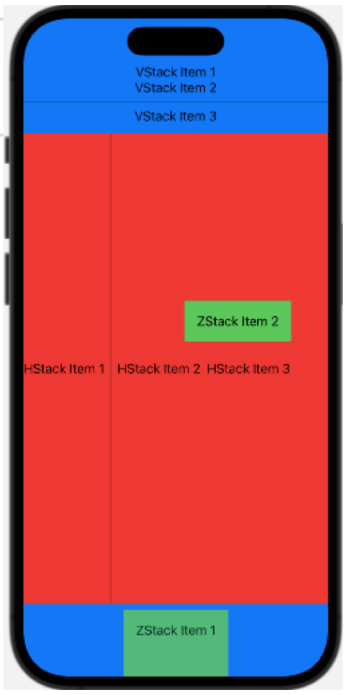
TheStacks

- You can also use the .frame modifier to adjust the width and height of a component

```
23                  Text("HStack Item 2")
24  //                Divider()
25  //                    .background(.black)
26  //                Spacer()
27                  Text("HStack Item 3")
28              }
29              .frame(
30                  maxWidth: .infinity,
31                  maxHeight: .infinity,
32                  alignment: .topLeading
33              )
34              .background(Color.red)
35              ZStack{
```

TheStacks

- .frame



77

Text

- A view that displays one or more lines of read-only text.
- Text("Hello World")

- **let** strMessage = ”This is a string variable"
- Text(strMessage)

- **let** intAge = 49
- Text("This is my age = \(myString)")

78

Text

- Text(”SRIMCA") .font(.title)

- Text(”Uka Tarsadia University")
- .font(.system(size: 12, weight: .light, design: .serif))
- .italic()

- Text(”Wrap a long string with width of 100 points")
- .frame(width: 100)

Text

- Text(”Uka Tarsadia University")
- .frame(width: 100)
- .lineLimit(1)

Text

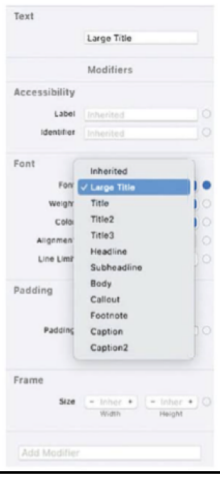
- Text(”Uka Tarsadia University")
- .frame(width: 100)
- .lineLimit(1)
- .truncationMode(.head)

Text

- Changing the Appearance of Text

```
Text("Large Title")
    .font(.largeTitle)
Text("Title")
    .font(.title)
Text("Title 2")
    .font(.title2)
Text("Title 3")
    .font(.title3)
Text("Caption")
    .font(.caption)
Text("Caption 2")
    .font(.caption2)
Text("Headline")
    .font(.headline)
Text("Subheadline")
    .font(.subheadline)
Text("Callout")
    .font(.callout)
Text("Footnote")
    .font(.footnote)
Text("Body")
    .font(.body)
```

Large Title
Title
Title 2
Title 3
Caption
Caption 2
Headline
Subheadline
Callout
Footnote
Body

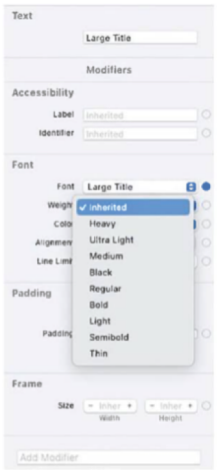


Text

- Weights for text

```
Text("Heavy")
    .font(.largeTitle)
    .fontWeight(.heavy)
Text("Ultra Light")
    .font(.largeTitle)
    .fontWeight(.ultraLight)
Text("Medium")
    .font(.largeTitle)
    .fontWeight(.medium)
Text("Black")
    .font(.largeTitle)
    .fontWeight(.black)
Text("Regular")
    .font(.largeTitle)
    .fontWeight(.regular)
Text("Bold")
    .font(.largeTitle)
    .fontWeight(.bold)
Text("Light")
    .font(.largeTitle)
    .fontWeight(.light)
Text("Semibold")
    .font(.largeTitle)
    .fontWeight(.semibold)
Text("Thin")
    .font(.largeTitle)
    .fontWeight(.thin)
```

Heavy
Ultra Light
Medium
Black
Regular
Bold
Light
Semibold
Thin

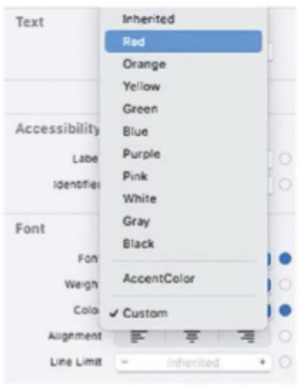


83

Text

- Color of the text

```
.foregroundColor(.red)
```



Dharmendra Bhatti 84

Label

- Label view can display both a string and an image side by side
- The Label view can use any image from Apple's free SF Symbols app
- https://developer.apple.com/sf-symbols/

Label

- Label("Text", systemimage: "SF Symbol image name here")
- Label("Lightning", systemImage: "bolt.fill")
- Label("Text", image: "image name here from Assets folder here")

Label

- if you want to customize the appearance of the text and/or image
- Label {
  Text("Alternate Label definition")
- } icon: {
  Image(systemName: "SF Symbol image name here")
- }

Label

- Adding a Border Around a Text or Label View
- Label(“border around a label view”, systemImage: “folder.fill”)
- .padding()
- .border(Color.green, width: 4)

Image

- A view that displays an image.

Image

- You can create images from following sources:
  - Image files in your app's **asset** library or bundle. Supported types include PNG, JPEG, HEIC, and more.
  - Instances of platform-specific image types, like **UIImage** and **NSImage**.
  - A bitmap stored in a Core Graphics **CGImage** instance.
  - System graphics from the **SF Symbols** set.

Image

- Image(”SRIMCA")
- .resizable()
- .aspectRatio(contentMode: .fit)
- Text(”My College")

Image

- Display an icon stored in the SF Symbols app
- Image(systemName: "hare.fill")

```
Image(systemName: "tortoise.fill")

Image(systemName: "tortoise.fill")
    .font(.largeTitle)

Image(systemName: "tortoise.fill")
    .font(.custom("", size: 46))
```

Image

- Clipping Images

```
Image("Cats")
    .resizable()
    .scaledToFill()
    .frame(width: 350, height: 350)
```

```
Image("Cats")
    .resizable()
    .scaledToFill()
    .frame(width: 350, height: 350)
    .clipShape(Circle())
```

93

Image

- .shadow(color: .red, radius: 46, x: 0, y: 0)
  - Color - Defines the shadow's color.
  - Radius - Defines the shadow's size around the view.
  - X - Defines the x (horizontal) offset of the shadow. A value of 0 centers the shadow in the horizontal direction around the view.
  - Y - Defines the y (vertical) offset of the shadow. A value of 0 centers the shadow in the vertical direction around the view.

Image

- Border around Image
- .overlay(Rectangle().stroke(Color.blue, linewidth: 10))

Image

- Faint/Sharp Image
- .opacity(0.75)

```
Image("goodCat")
    .resizable()
    .scaledToFill()
    .frame(width: 250, height: 250)
    .opacity(0.75)

Image("goodCat")
    .resizable()
    .scaledToFill()
    .frame(width: 250, height: 250)
    .opacity(0.5)

Image("goodCat")
    .resizable()
    .scaledToFill()
    .frame(width: 250, height: 250)
    .opacity(0.25)
```

Button

- A control that initiates an action.
  - To create a button, you need to define
  - The title that displays text on the button Swift code that runs when the user taps the button

- Button("Click here") {
- // code to run
- }

Button

- A second way to create a button gives you more flexibility in modifying the appearance of the text

- Button {
  // code to run
- } label: {
  Text("Click here")
- }

Button

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

- Button(action: signIn) {
- Label("Sign In", systemImage: "arrow.up")
- }

Button

- Button("Sign In", systemImage: "arrow.up", action: signIn)

Button

- Styles
  - .buttonStyle(.plain)
  - .buttonStyle(.bordered)
  - .buttonStyle(.borderedProminent)
  - .buttonStyle(.borderless)

Button

```
Button ("Plain text button") {
    // code to run
}

Button {
    // code to run
} label: {
    Text("Custom text button")
        .font(.largeTitle)
        .foregroundColor(.green)
        .padding()
        .border(Color.red, width: 6)
}

Button {
    // code to run
} label: {
    Label("Image button", systemImage: "hare.fill")
        .font(.largeTitle)
        .foregroundColor(.purple)
        .padding()
        .border(Color.blue, width: 6)
}

Button {
    // code to run
} label: {
    Image("kitty")
        .resizable()
        .scaledToFill()
        .frame(width: 150, height: 150)
        .clipShape(Circle())
        .overlay(Circle().stroke(Color.green, lineWidth: 8))
}
```

Plain text button

Custom text button

Image button

Button

- Icon and Text Button

- Button(action: {
- // Action to perform
- }) {
- // Custom view for the button
- HStack {
- Image(systemName: "square.and.arrow.down.fill")
- Text("Download")
- }
- }

Button

- Show/hide detail text

```
@State private var showDetails = false

var body: some View {
    VStack(alignment: .center) {
        Button("Show details") {
            showDetails.toggle()
        }

        if showDetails {
            Text("Visit https://utu.ac.in")
                .font(.largeTitle)
        }
    }
}
```

Button – RollDice Example

- On tap of button, roll dice and display result

```
@State var randomNumber = 0
var body: some View {
    VStack (spacing: 28) {
        Text("Random number = \(randomNumber)")
        Button("Roll dice") {
            let firstDie = Int.random(in: 1...6)
            let secondDie = Int.random(in: 1...6)
            randomNumber = firstDie + secondDie
        }
    }
}
```

Button – RollDice Examp

- On tap of button,
- roll dice and display result

Dharmendra Bhatti

Button – RollDice Example

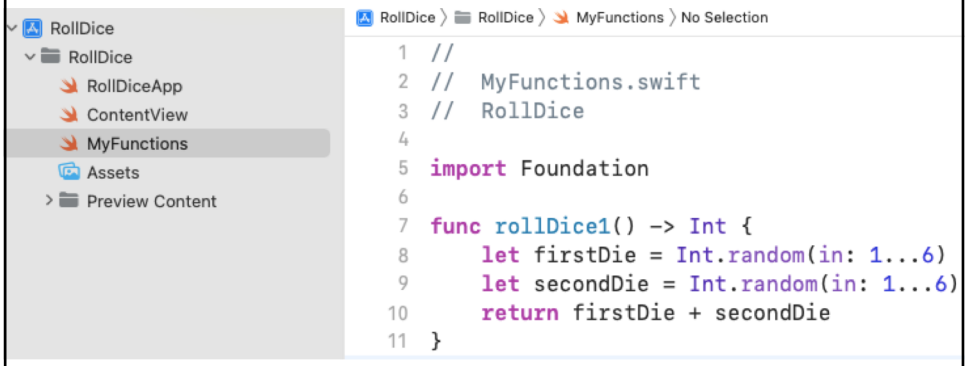
- Define function in the same structure

```
struct ContentView: View {
    @State var randomNumber = 0
    var body: some View {
        VStack (spacing: 28) {
            Text("Random number = \(randomNumber)")
            Button("Roll dice") {
                let firstDie = Int.random(in: 1...6)
                let secondDie = Int.random(in: 1...6)
                randomNumber = rollDice()
            }
        }
    }
    func rollDice() -> Int {
        let firstDie = Int.random(in: 1...6)
        let secondDie = Int.random(in: 1...6)
        return firstDie + secondDie
    }
```

107

Button – RollDice Example

- Define function in separate file



Dharmendra Bhatti 108

Segmented Control

- The main idea behind a segmented control is to display two or more options in a condensed space rather than use multiple buttons.

Segmented Control

- To create a segmented control, you need the following:
  - A State variable to represent which segment (option) tire user chose
  - A Picker view that lists two or more options
  - A tag property linked to each option
  - The SegmentedPickerStyle modifier

Segmented Control – Example1

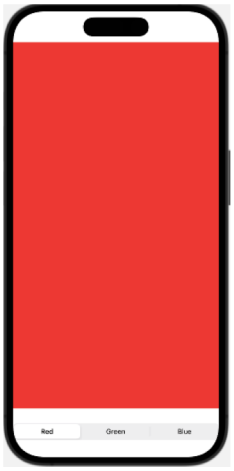
- Picker("", selection: $selectedItem) {
- Text("Fish")
- Text("Tortoise")
- Text("Hare")
- Text("Bird")
- }.pickerStyle(.segmented)

Segmented Control – Example1

- Without pickerStyle(.segmented)

Segmented Control – Example1

- With pickerStyle(.segmented)



Dharmendra Bhatti 113

Segmented Control – Example2

- .onChange

```
@State private var subject = "Select Subject"

var body: some View {
    VStack (spacing: 16) {
        Text(subject)

        Picker("", selection: $subject) {
            Text("701").tag("ML")
            Text("702").tag("iOS")
            Text("703").tag("Wordpress")
        } .pickerStyle(.segmented)
            .background(.red)
            .onChange(of: subject) { newValue in
                switch newValue {
                case "ML" :
                    subject = "Machine Learning"
                case "iOS" :
                    subject = "Mobile Application Development
                        with iOS"
                case "Wordpress" :
                    subject = "Content Management System using
                        Wordpress"
                default:
                    break
                }
            }
    }
}
```

Segmented Control - Example3

```
@State private var subject = "Select Subject"
let subjectArray = ["ML", "iOS", "Wordpress"]

var body: some View {
    VStack (spacing: 16) {
        Text(subject)

        Picker("", selection: $subject) {
            ForEach(subjectArray, id: \.self) {
                Text($0)
            }
        } .pickerStyle(.segmented)
            .background(.red)
            .onChange(of: subject) { newValue in
                switch newValue {
                case "ML" :
                    subject = "Machine Learning"
                case "iOS" :
                    subject = "Mobile Application Development
                        with iOS"
                case "Wordpress" :
                    subject = "Content Management System using
                        Wordpress"
                default:
                    break
                }
            }
}
```

Text Field

- TextField
- SecureField
- TextEditor

Text Field

- A Text Field lets the user type in a single line of text such as a name or an address.
- Optionally, Text Fields can display placeholder text that appears in light gray and is used to explain what type of information the Text Field expects.

Secure Field

- A Secure Field works exactly like a Text Field except that it masks any text the user types in.
- That can be useful when asking the user to type in sensitive information such as pin, password, or credit card numbers.

Text Editor

- A Text Editor appears as a large box where the user can type and edit several lines of text such as multiple paragraphs.

Text Field

- Since Text Fields, Secure Fields, and Text Editors need to store data, they need to work with a State variable that can hold a String data type such as

- @State **private var** message = ""

```
TextField("Placeholder text", text: $message)
```

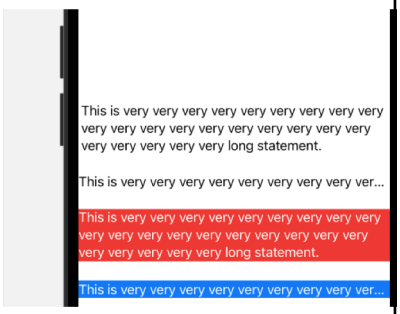
Text Field

- Expandable Text Field

```
@State var message = "This is very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very very long statement."
var body: some View {
    VStack (spacing: 24) {
        Text(message)

        TextField("Text Field without axis", text: $message)

        TextField("Vertical", text: $message, axis: .vertical)
            .background(Color.red)
            .foregroundColor(Color.white)
        TextField("Horizontal", text: $message, axis: .horizontal)
            .background(Color.blue)
            .foregroundColor(Color.white)
    }
}
```



Dharmendra Bhatti 121

Text Field

- .lineLimit(3)
- .lineLimit(2…5)
- .textFieldStyle(.roundedBorder)
- .disableAutocorrection(true)
- .textContentType(.emailAddress)

Dharmendra Bhatti 122

Text Field

- .keyboardType(.emailAddress)

123

Text Field

- .submitLabel(.done)
  - .continue - Adds a Continue button
  - .done - Adds a Done button
  - .go - Adds a Go button
  - .join - Adds a loin button
  - .next - Adds a Next button
  - .return - Adds a Return button
  - .route - Adds a Route button
  - .search - Adds a Search button
  - .send - Adds a Send button

124

Text Editor

- A Text Editor lets the user type in multiple lines of text
- It expands to fill all available space.
- It's best to use the .frame modifier to define a specific size for the Text Editor.

Text Editor

- So if you want to hide a virtual keyboard when using a Text Editor, you need to do the following:
  - Create a Focus State variable that represents a Boolean value.
  - Add the .focused modifier to the Text Editor and use the Focus State variable.
  - Create an additional control, such as a button, that sets the Focus State variable to false.

Text Editor

- .focused($dismissKeyboard)
- .frame(width: 250, height: 50)

```
@State var message = "Write text here..."
@FocusState var dismissKeyboard: Bool

var body: some View {
    VStack (spacing: 24) {
        TextEditor(text: $message)
            .focused($dismissKeyboard)
            .frame(width: 250, height: 50)
        Button("Hide Keyboard") {
            dismissKeyboard = false
        }
    }
}
```

127

Picker

- A Picker displays a list of options defined by multiple Text views.

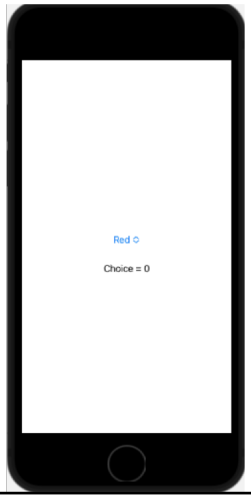
```
@State var choice = 0

var body: some View {
    VStack (spacing: 24) {
        Picker("", selection: $choice) {
            Text("Red").tag(1)
            Text("Green").tag(2)
            Text("Blue").tag(3)
            Text("Yellow").tag(4)
            Text("Orange").tag(5)
        }
        Text("Choice = \(choice)")
    }
```

128

Picker

- A Picker displays a list of options defined by multiple Text views.



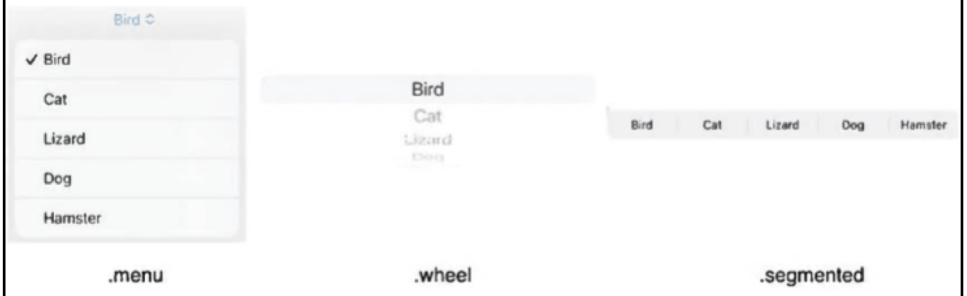
Picker

- By default, a Picker view appears as a menu.

- .pickerStyle(.menu)
- .pickerStyle(.wheel)
- .pickerStyle(.segmented)

Picker

- By default, a Picker view appears as a menu.



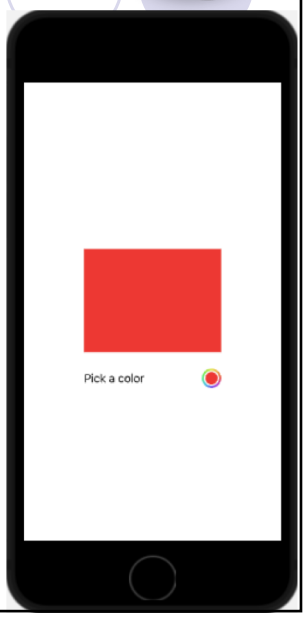
Color Picker – Example1

- @State **var** myColor = Color.red
- **var** body: **some** View {
- Vstack {
- Rectangle()
- .frame(width: 200, height: 150)
- .foregroundcolor(myColor)
- ColorPicker("Pick a color", selection: $myColor)

Color Picker – Example1

- Pick a color

```
@State var myColor = Color.red
var body: some View {
    VStack (spacing: 24) {
        Rectangle()
            .frame(width: 200, height: 150)
            .foregroundColor(myColor)
        ColorPicker("Pick a color",
            selection: $myColor)
            .frame(width: 200)
    }
}
```



Dharmendra Bhatti

Date Picker

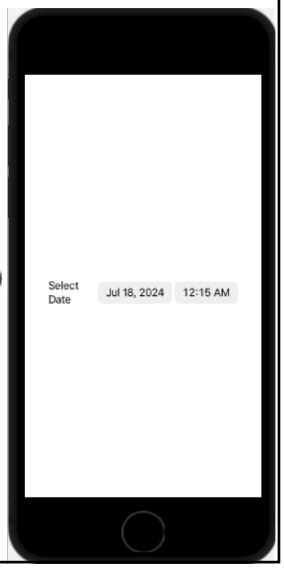
- Instead of typing a date or time, users can simply click the date or time they want
- DatePicker(selection: $myDate, label: { Text("Date") })

Dharmendra Bhatti 134

Date Picker

- Date Selection

```
@State var myDate = Date.now
var body: some View {
    VStack (spacing: 24) {
        DatePicker(selection: $myDate,
                   label: {Text("Select Date")
            .frame(width: 300)
    }
}
```



Dharmendra Bhatti

Date Picker

- .datePickerStyle(.graphical)

Dharmendra Bhatti 136

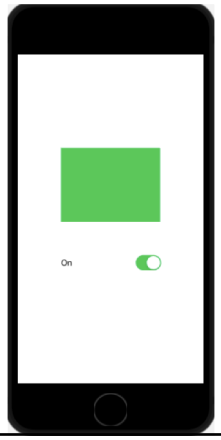
Date Picker

- @State **var** dates = Set<DateComponents>()
- **var** body: **some** View {
- VStack {
  MultiDatePicker("Select Dates", selection: $dates)
- }

Dharmendra Bhatti 137

Toggles

- A Toggle gives users exactly two choices such as on or off, yes or no, and true or false.



138

Toggles

- On/Off

```
@State var myToggle = true
var body: some View {
    VStack {
        Rectangle()
            .frame(width: 200, height: 150)
            .foregroundColor(myToggle ? .green : .red)
        Toggle(myToggle ? "On" : "Off", isOn: $myToggle)
            .frame(width: 200, height: 150)
    }
}
```

Dharmendra Bhatti 139

Stepper

- A Stepper restricts user input to a range of valid data.
- Steppers display a minus/plus icon that users can click to increment/decrement a value by a fixed amount, up or down.

Dharmendra Bhatti 140

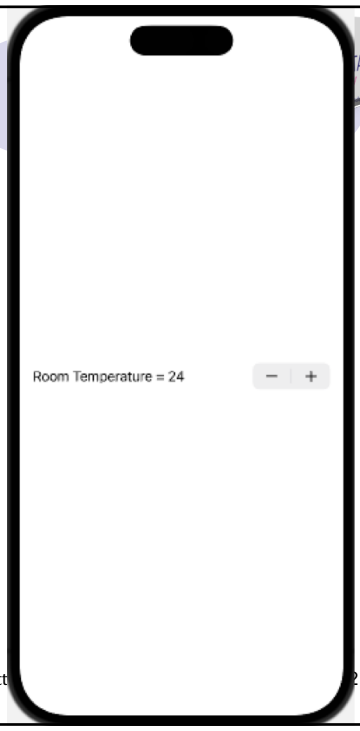
Stepper

- Room Temperature

```
@State var roomTemperature = 24
var body: some View {
    VStack {
        Stepper(value: $roomTemperature, in: 18...30) {
            Text("Room Temperature = \(roomTemperature)")
        }
        .padding()
    }
}
```


Stepper

- Room Temperature



Slider

- Sliders let users drag to input a specific value without any typing whatsoever.

143

Slider

```
@State var sliderValue = 1.0
var body: some View {
    VStack (spacing: 24) {
        Text("Slider value = \(sliderValue)")

        Slider(value: $sliderValue, in: 1...5, step: 1) {
            Text("Slider")
        } minimumValueLabel: {
            Text("1")
        } maximumValueLabel: {
            Text("5")
        }
        .padding()
        .accentColor(.red)
```

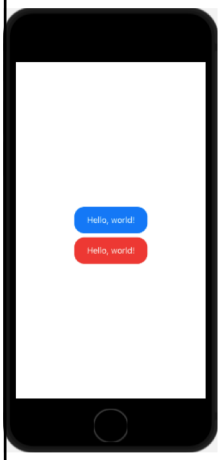
Dharmendra Bhatti

144

ViewModifier

- Apply group of styles using ViewModifier
- Create a custom modifier that adds rounded corners and a background to a **Text** view

View Modifier




```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, world!")
                .modifier(BackgroundStyle(bgColor: .blue))
            Text("Hello, world!")
                .backgroundStyle(color: .red)
        }
    }
}
struct BackgroundStyle: ViewModifier {
    var bgColor: Color
    func body(content: Content) -> some View{
        content
            .frame(width:UIScreen.main.bounds.width * 0.3)
            .foregroundStyle(.white)
            .padding()
            .background(bgColor)
            .cornerRadius(20)
    }
}
extension View {
    func backgroundStyle(color: Color) -> some View{
        self.modifier(BackgroundStyle(bgColor: color))
    }
}
```

ViewBuilder

- ViewBuilder is a custom parameter attribute that constructs views from closures
- @ViewBuilder allows to create custom views that can build and combine other views to create complex layouts.

ViewBuilder

- BlueCircle.swift

```
struct BlueCircle<Content: View>: View {
    let content: Content
    init(@ViewBuilder content: () -> Content) {
        self.content = content()
    }
    var body: some View {
        HStack {
            content
            Spacer()
            Circle()
                .fill(Color.blue)
                .frame(width:20, height:30)
        } .padding()
    }
}
```

ViewBuilder

- Use custom view

```
var body: some View {
    VStack {
        BlueCircle {
            Text("some text here")
            Rectangle()
                .fill(Color.red)
                .frame(width: 40, height: 40)
        }
        BlueCircle {
            Text("Another example")
        }
```

Dharmendra Bhatti 149

ViewBuilder

- Use custom view

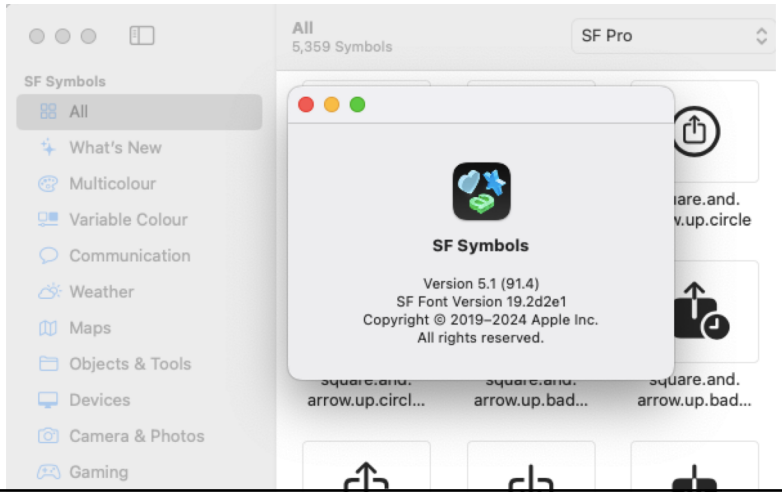
Dharmendra Bhatti

SF Symbols

- SF Symbols is a library of symbols that are designed to integrate seamlessly with San Francisco, the system font for Apple platforms.

SF Symbols

- 5359 symbols in version 5.1



SF Symbols

- Image(systemName:)

```
VStack (spacing: 24) {
    HStack{
        Image(systemName: "s")
        Image(systemName: "r")
        Image(systemName: "i")
        Image(systemName: "m")
        Image(systemName: "c")
        Image(systemName: "a")
    }
    .symbolVariant(.fill.circle)
    .foregroundStyle(.yellow, .red)
    .font(.largeTitle)
```

Integrating UIKit into SwiftUI

- We can display UIKit views in SwiftUI by using the UIViewRepresentable protocol.

Integrating UIKit into SwiftUI

- Add ActivityIndicator.swift in SwiftUI project

```
import SwiftUI
struct ActivityIndicator: UIViewRepresentable {
    var animating: Bool
    func makeUIView(context: Context) -> UIActivityIndicatorView {
            return UIActivityIndicatorView()
    }
    func updateUIView(_ activityIndicator: UIActivityIndicatorView,
                      context: Context) {
        if animating {
            activityIndicator.startAnimating()
        } else {
            activityIndicator.stopAnimating()
        }
```

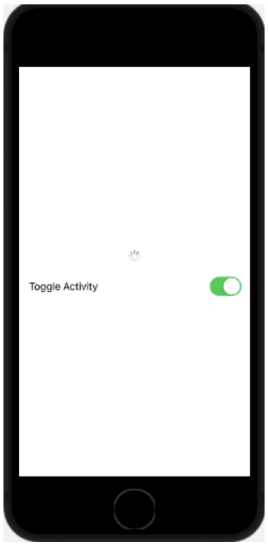
Integrating UIKit into SwiftUI

- Use ActivityIndicator view in ContentView

```
@State private var animate = true
var body: some View {
    VStack {
        ActivityIndicator(animating:  animate)
        HStack {
            Toggle(isOn: $animate) {
                Text("Toggle Activity")
            }
        }
        .padding()
```

Integrating UIKit into SwiftUI

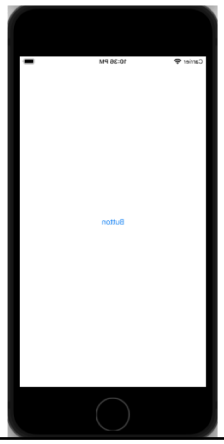
- On/Off



157

Add SwiftUI to legacy UIKit App

- Navigate from a UIKit view to a SwiftUI view while passing a text to SwiftUI view

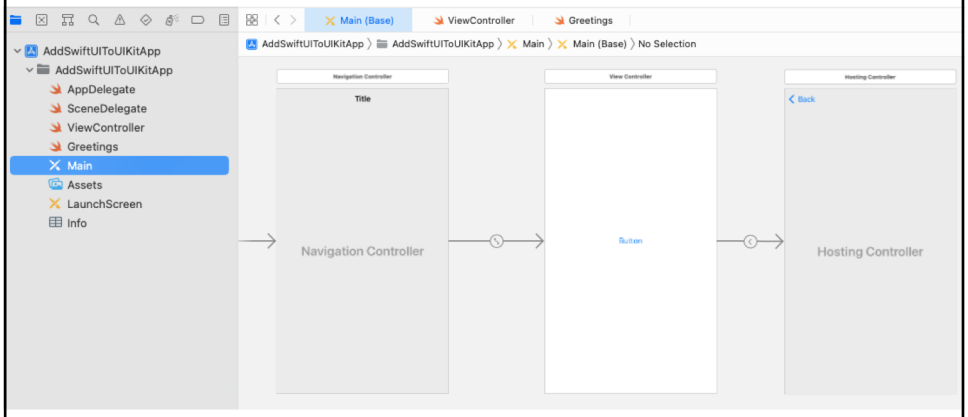


Dharmendra Bhatti

158

Add SwiftUI to legacy UIKit App

- Main



Add SwiftUI to legacy UIKit App

- ViewController.swift

```swift
import UIKit
import SwiftUI
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }
    @IBSegueAction func GoToSwiftUI(_ coder: NSCoder) -> UIViewController? {
        let greetings = "Hello From UIKit"
        let rootView = Greetings(textFromUIKit: greetings)
        return UIHostingController(coder: coder, rootView: rootView)
    }
}
```

Add SwiftUI to legacy UIKit App

- Greetings.swift (SwiftUI Code)

```
import SwiftUI

struct Greetings: View {
    var textFromUIKit: String
    var body: some View {
        Text(textFromUIKit)
    }
}
```


Questions ???