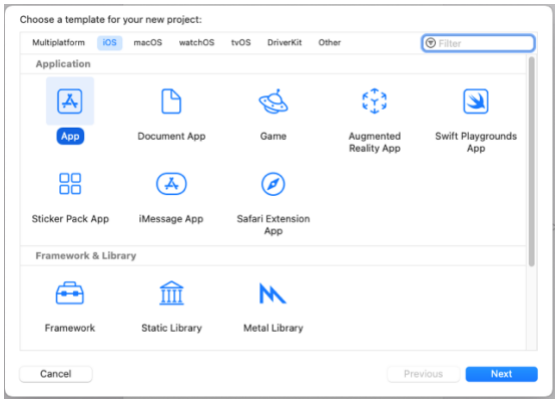
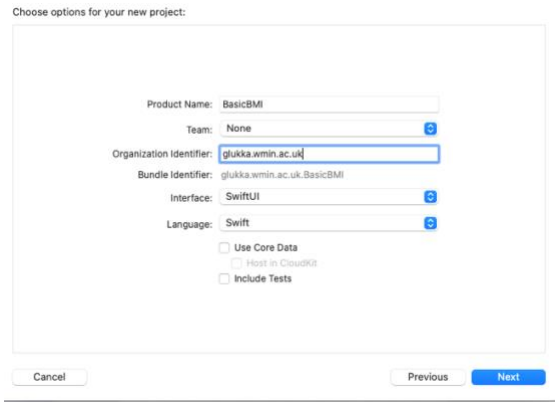
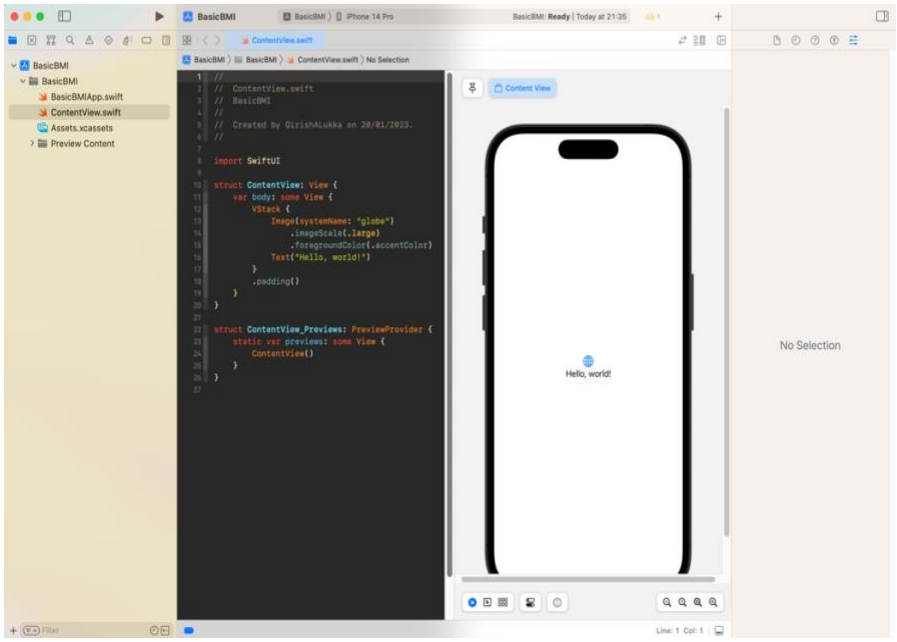
Week 1 SwiftUI Tutorial 6COSC021W &7SENG002W

To get familiar with SwiftUI application development and XCode 14 let us build a basic BMI Calculator. The requirements for this app are simple – get user height, weight, calculate BMI and display it with a suitable message.

Launch XCode and select iOS App template and interact with pop-ups to create a BasicBMI (Body Mass Index) app as shown below:







Create a label using Text view.
Create two TextField Views for weight and height inputs. There are other ways to get user inputs, but here use TextFields.
Embed all these in a VStack:

Modify ContentView – add the code shown below:

```
struct ContentView: View {
    @State private var weightText: String = ""
    @State private var heightText: String =
    ""

    var body: some View
        { VStack
            {
             Text("BMI    Calculator:").font(.largeTitle)
             TextField("Enter Weight (in kilograms)",text: $weightText)
                  .textFieldStyle(RoundedBorderTextFieldStyle())
                  .border(Color.black)

             TextField("Enter Height (in meters)",text: $heightText)
                  .textFieldStyle(RoundedBorderTextFieldStyle())
                  .border(Color.black)

        }.padding()
    }
}
```

Code Explanation

```
TextField("Enter Weight (in kilograms)",text: $weightText)
```

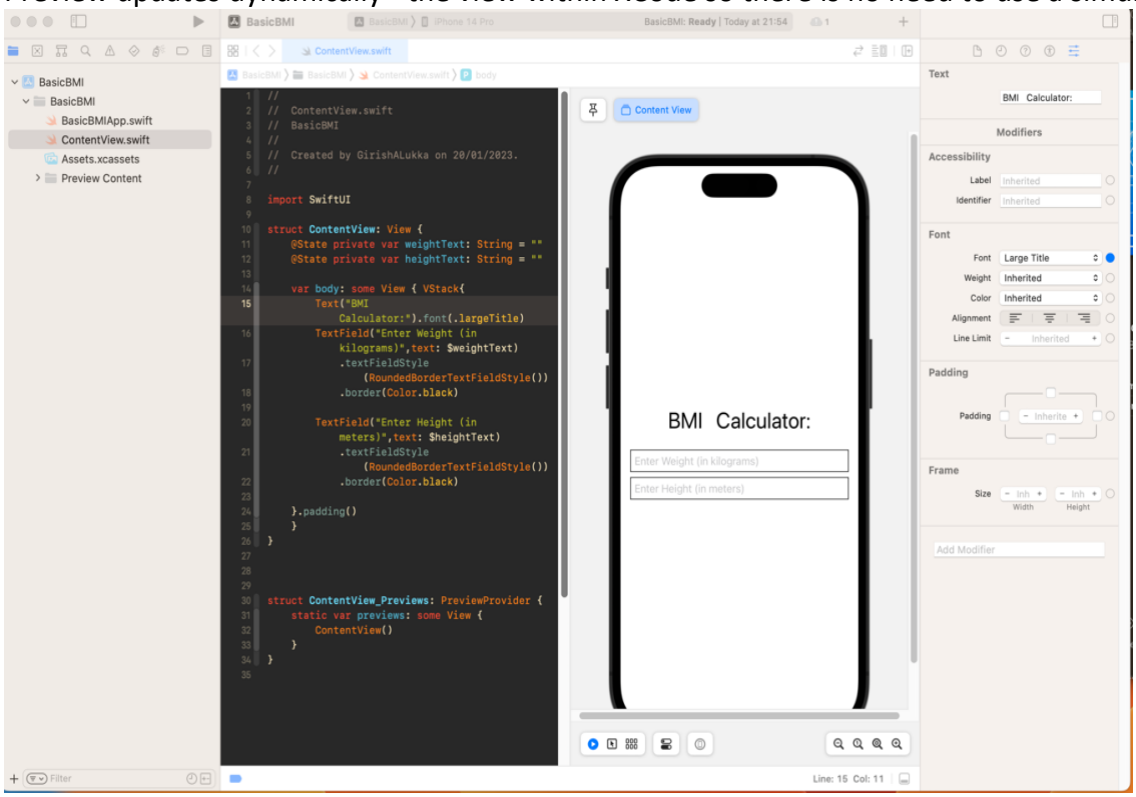
The first argument to the TextField view is the placeholder that provides hints to the user on how to enter input. In this field, the user must enter weight in kilograms e.g., 85 (meaning 85 kg).

Two private state variables weightText and heightText are bound to a TextField object through the text parameter.

```
TextField("Enter Weight (in kilograms)",text: $weightText)
TextField("Enter Height (in meters)",text: $heightText)
```

Note use $ prefix to bind the state variable to the view.

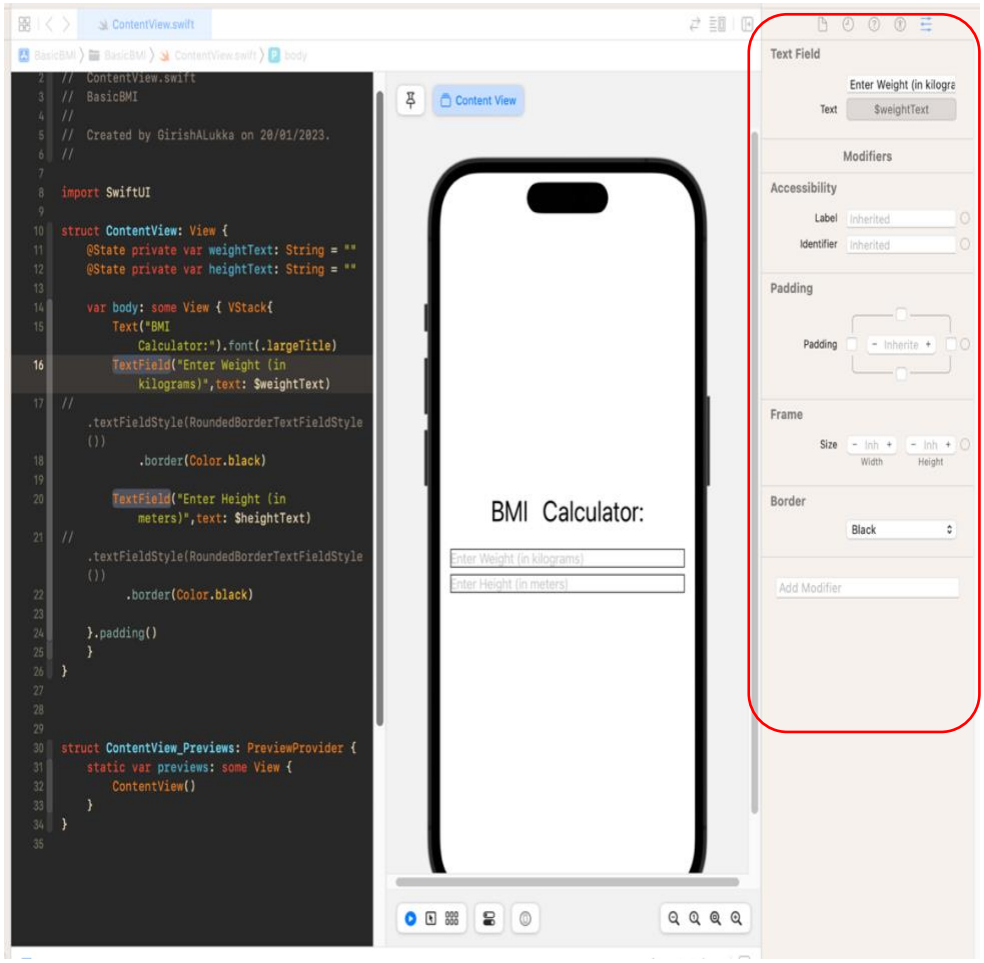
Preview updates dynamically - the view within XCode so there is no need to use a simulator.



Examine the effect of commenting out the TextField modifiers.

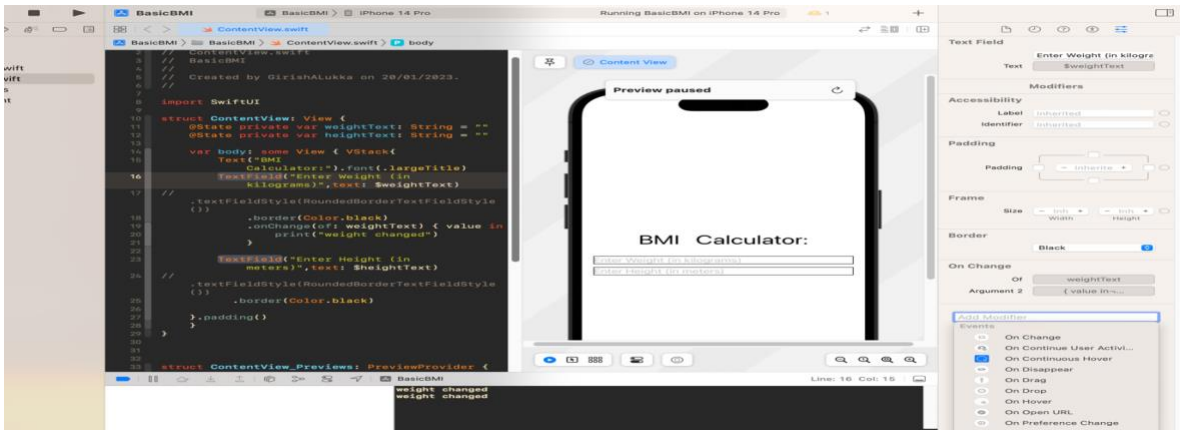
```
    var body: some View { VStack{
        Text("BMI    Calculator:").font(.largeTitle)
        TextField("Enter Weight (in kilograms)",text: $weightText)
//            .textFieldStyle(RoundedBorderTextFieldStyle())
            .border(Color.black)

        TextField("Enter Height (in meters)",text: $heightText)
//            .textFieldStyle(RoundedBorderTextFieldStyle())
          .border(Color.black)

    }.padding()
    }
}
```

Highlight the Weight TextField by hovering the mouse over and see the information in the properties panel.

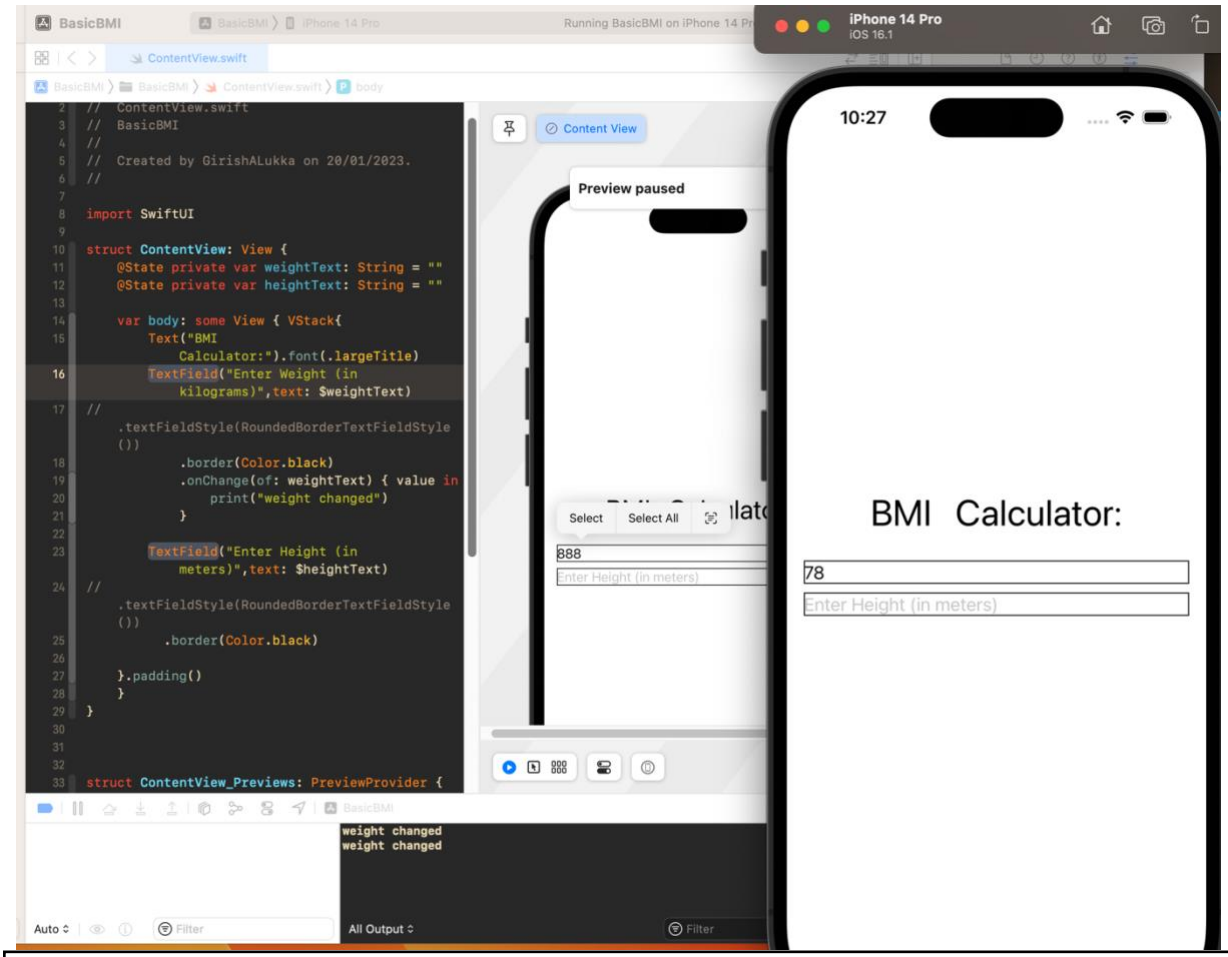


Click in the Add Modifier tab and examine the options, try some of them later and see how the code changes as well as the preview.

Just for demo here, I have added onChange event to Weight TextField



Next identify the change and action to follow.
Here I have used the value entered in the TextField to trigger a message in the console (need to use a simulator to see the message, preview does not support console messages) as shown below.




```
TextField("Enter Weight (in kilograms)",text: $weightText)
                .textFieldStyle(RoundedBorderTextFieldStyle())
                .border(Color.black)
```

```
            .onChange(of: weightText) { value in
                print("weight changed")              }
```

Next add a button that when clicked will calculate the BMI and display it to the user.

List the events and actions that must take place for the button to be effective.

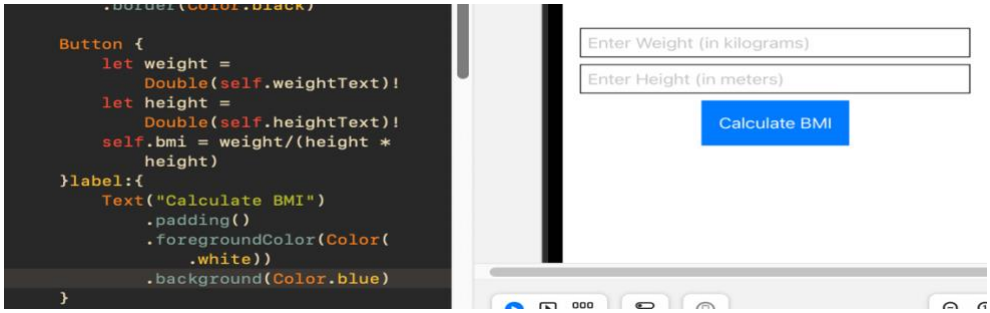
First add a variable to store the calculated bmi:

@State private var bmi: Double = 0

```
@State private var weightText: String = ""
@State private var heightText: String = ""
@State private var bmi: Double = 0
```

Next add the button, add it to the VStack after the TextFields and a Text View to display the result:

```
Button {
                        -    - - -
                        -    - - -
            let weight = Double(self.weightText)!
            let height = Double(self.heightText)!
            self.bmi = weight/(height * height)
        }label:{
            Text("Calculate BMI")
                .padding()
                .foregroundColor(Color(.white))
                .background(Color.blue)
        }
        Text("BMI:   \(bmi))
            .font(.title)
            .padding()
```

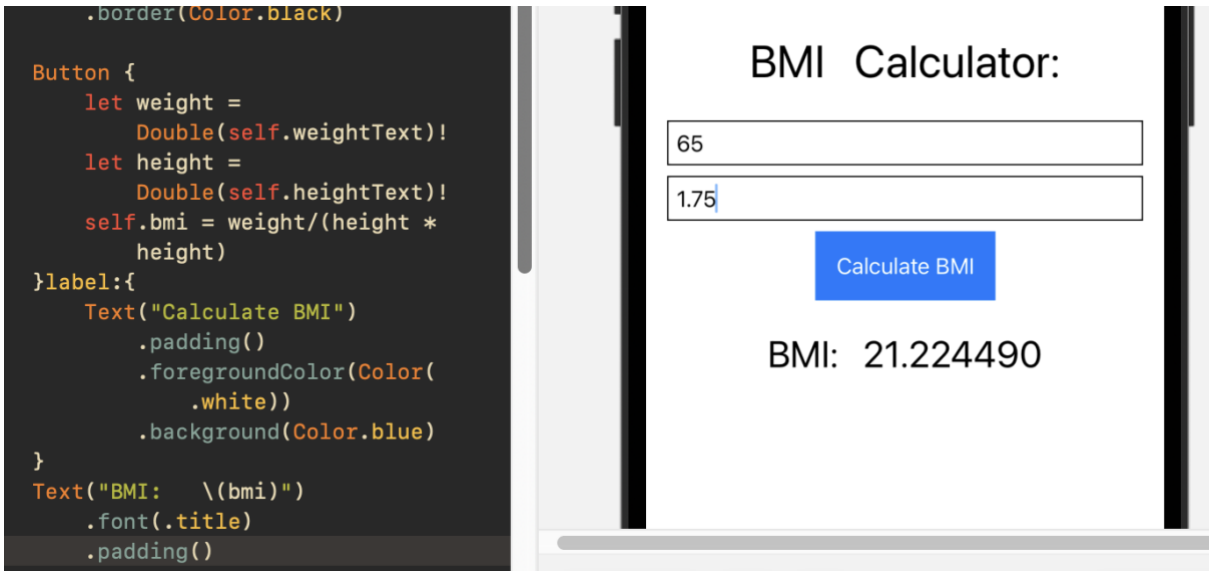


Note that user input values which are String data has been converted to Double.
The exclamation mark "!" is about "Optionals" and here we are forced into unwrapping optional.

Aside:
Forced Unwrapping Optionals
Forced unwrapping extracts the value of the optional variable. Forced unwrapping **assumes the value is definitely not-nil**. Thus, it throws a fatal error if the variable is nil. Optionals can be forced unwrapped by placing ! after the variable name.

Forced unwrapping is a "quick and dirty" coding practice and should be avoided otherwise the app could crash.

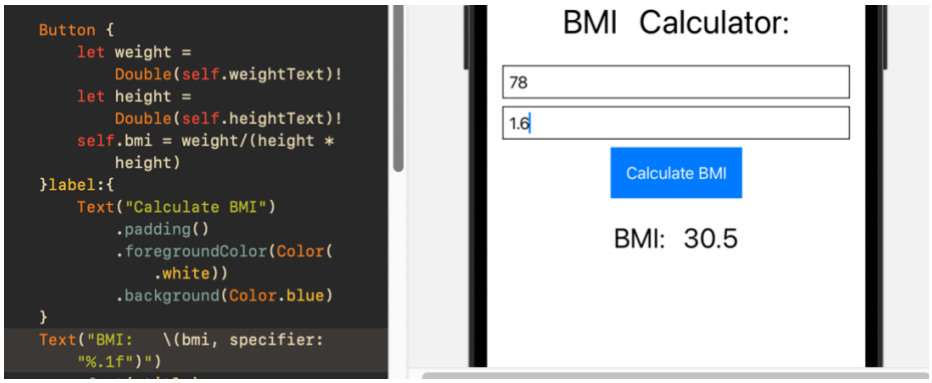
Test app iin a simulator – select iPhone 14 Pro and test the app.



What happens if you press calculate BMI button with no data for weight or if it is non-numeric?

To display the result to 1 decimal place, add a specifier parameter to the string interpolation:

```
Text("BMI: \(bmi, specifier: "%.1f")")
.font(.title)
.padding()
```



Next, classify the BMI value as below and display it to the user:

Underweight: Your BMI is less than 18.5.
Healthy weight: Your BMI is 18.5 to 24.9.
Overweight: Your BMI is 25 to 29.9.

Create another variable to store the classification:

```
@State private var classification: String = ""
```

Modify the Button code and the display:

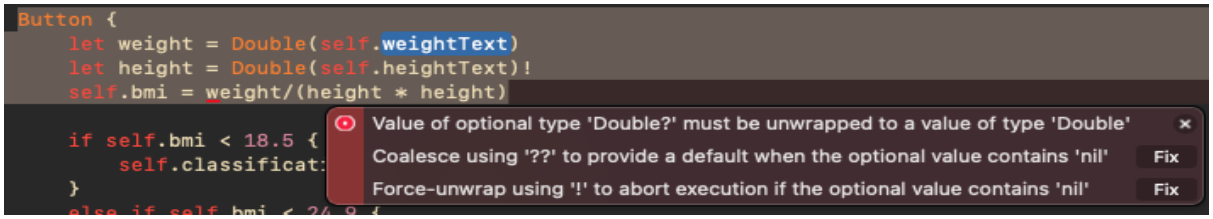
```
Button {
        let weight = Double(self.weightText)!
        let height = Double(self.heightText)!
        self.bmi = weight/(height * height)

        if self.bmi < 18.5 {
            self.classification = "Underweight"
        }
        else if self.bmi < 24.9 {
            self.classification = "Healthy weight"
        }
        else if self.bmi < 29.9 {
            self.classification = "Overweight"
```

```
            }
            else {
                self.classification = "Obese"
            }
        }label:{
            Text("Calculate BMI")
                .padding()
                .foregroundColor(Color(.white))
                .background(Color.blue)
        }
        Text("BMI:  \(bmi, specifier: "%.1f"), \(classification)")
            .font(.title)
            .padding()
```

Test your app

What happens if the ! mark is removed from, say the weight constant in the button code?

```
let weight = Double(self.weightText)
```



The code fails to compile and reports an error with suggestions how to fix the error – an option we used was forced unwrapping.

One correct approach is to unwrap the optional using ‘if let’ which checks that optional variable contains an actual value and bind the non-optional form to a temporary variable. This is a safe way to “unwrap” optionals.

Modify Button code:

```
Button {
            var weight: Double = 0
            var height: Double  = 0

            if let weightDouble = Double(self.weightText) {
                weight = weightDouble
            }
            if let heightDouble = Double(self.heightText) {
                height = heightDouble
            }

            self.bmi = weight/(height * height)
```

Code change effect - only move forward if *Double(self.weightText)* is not nil (i.e. it's a valid double numerical value) and in such a case, assign the value double to temporary variable *weightDouble* (*weightDouble* does not exist outside the scope). Then, assign *weightDouble* to *weight*.

Test your app with 3 cases as below and note the results:

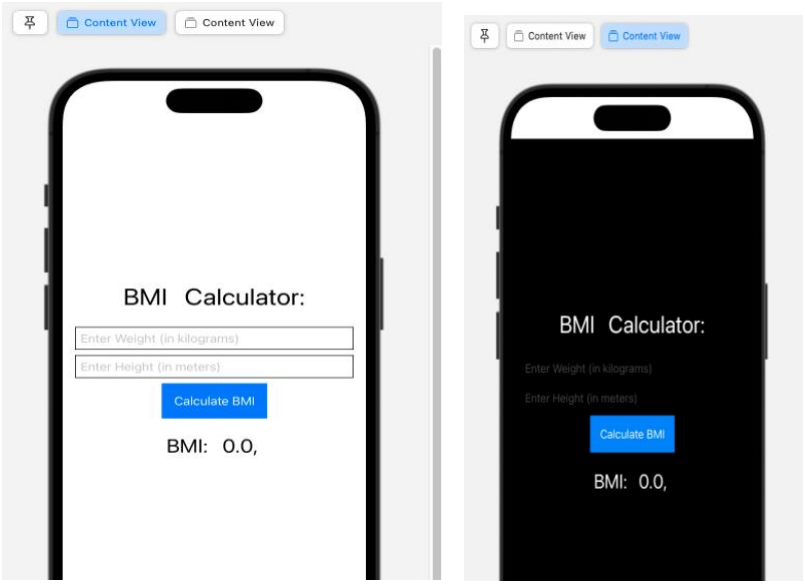
1. Press Calculate with no data
2. Weight 1, height blank
3. Weight blank, height 1

The results show that the app whilst it is not crashing, is far from working as it should.

To see the app in multiple modes, edit ContentView_Previews:

```
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
        ZStack{
```

```
        Color(.black)
        ContentView().environment(\.colorScheme, .dark)
    }
}
}
```



This tutorial ends here.