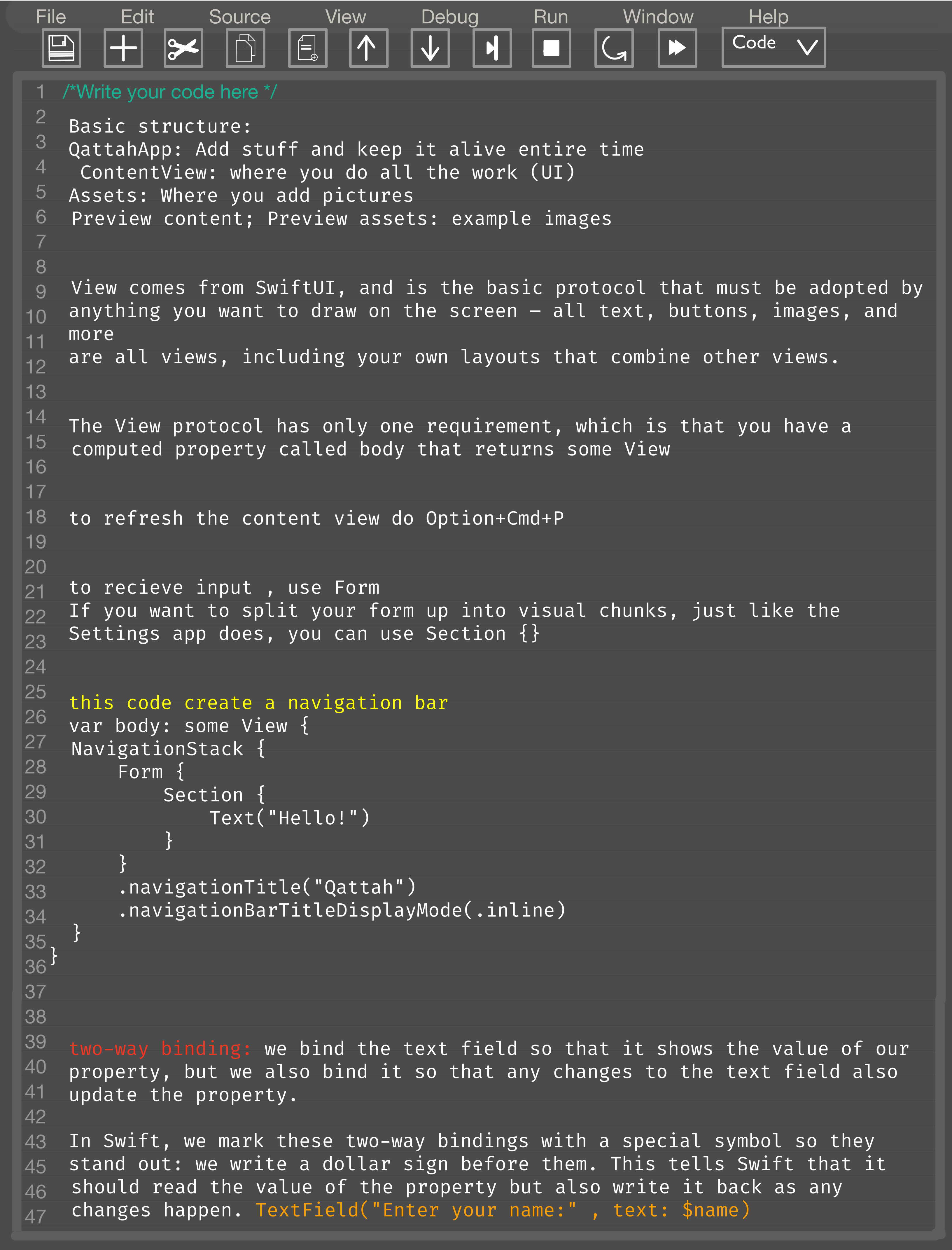
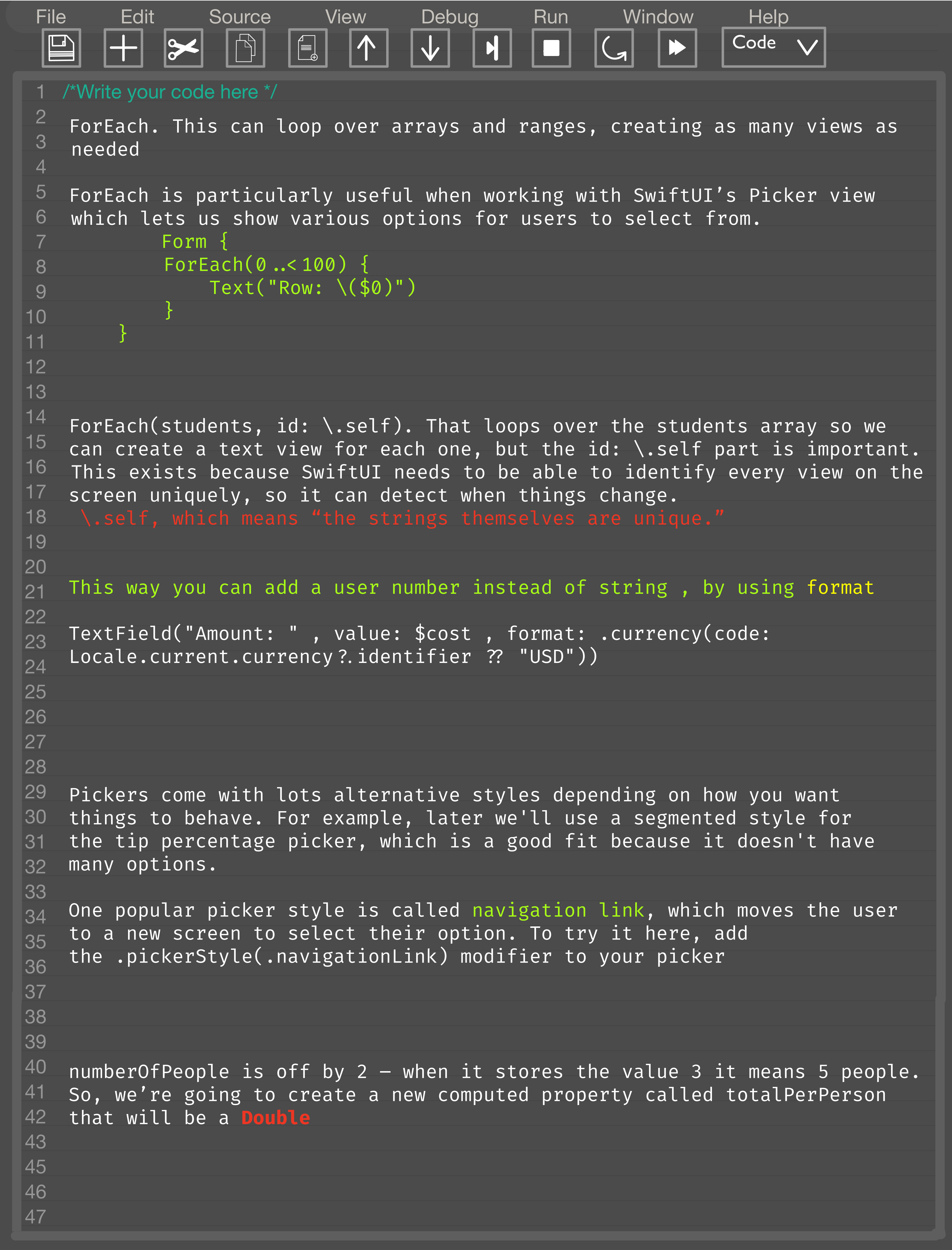


```
/*Write your code here */
 Basic structure:
 QattahApp: Add stuff and keep it alive entire time
  ContentView: where you do all the work (UI)
 Assets: Where you add pictures
 Preview content; Preview assets: example images


 View comes from SwiftUI, and is the basic protocol that must be adopted by
 anything you want to draw on the screen – all text, buttons, images, and
 more
 are all views, including your own layouts that combine other views.


 The View protocol has only one requirement, which is that you have a
 computed property called body that returns some View


 to refresh the content view do Option+Cmd+P


 to recieve input , use Form
 If you want to split your form up into visual chunks, just like the
 Settings app does, you can use Section {}


 this code create a navigation bar
 var body: some View {
 NavigationStack {
     Form {
         Section {
             Text("Hello!")
         }
     }
     .navigationTitle("Qattah")
     .navigationBarTitleDisplayMode(.inline)
 }
}


 two-way binding: we bind the text field so that it shows the value of our
 property, but we also bind it so that any changes to the text field also
 update the property.

 In Swift, we mark these two-way bindings with a special symbol so they
 stand out: we write a dollar sign before them. This tells Swift that it
 should read the value of the property but also write it back as any
 changes happen. TextField("Enter your name:" , text: $name)
```



```
/*Write your code here */
```

ForEach. This can loop over arrays and ranges, creating as many views as needed

ForEach is particularly useful when working with SwiftUI's Picker view which lets us show various options for users to select from.

```
        Form {
        ForEach(0..<100) {
            Text("Row: \($0)")
        }
    }
```

ForEach(students, id: \.self). That loops over the students array so we can create a text view for each one, but the id: \.self part is important. This exists because SwiftUI needs to be able to identify every view on the screen uniquely, so it can detect when things change.
 \.self, which means "the strings themselves are unique."

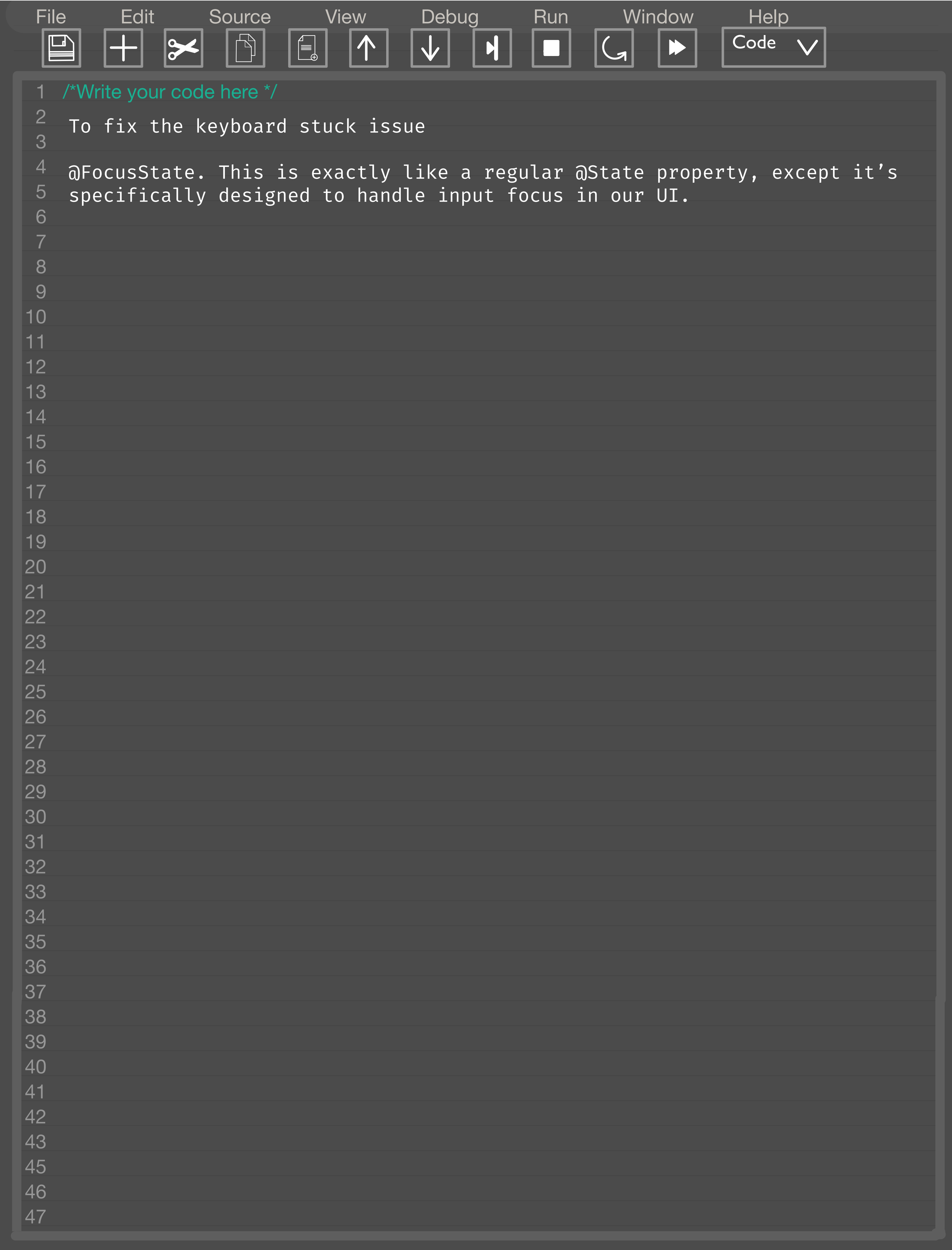
This way you can add a user number instead of string , by using format

```
TextField("Amount: " , value: $cost , format: .currency(code: Locale.current.currency?.identifier ?? "USD"))
```

Pickers come with lots alternative styles depending on how you want things to behave. For example, later we'll use a segmented style for the tip percentage picker, which is a good fit because it doesn't have many options.

One popular picker style is called navigation link, which moves the user to a new screen to select their option. To try it here, add the .pickerStyle(.navigationLink) modifier to your picker

numberOfPeople is off by 2 – when it stores the value 3 it means 5 people. So, we're going to create a new computed property called totalPerPerson that will be a **Double**



```
/*Write your code here */
 To fix the keyboard stuck issue

 @FocusState. This is exactly like a regular @State property, except it’s
 specifically designed to handle input focus in our UI.
```