

# Développement iOS

## Deuxième journée

Ezequiel GOMES 24/04/2025

# Tables des matières

Journée du 24/04/2025

- Les fonctions
- Les Structs
- Les Protocols
- Création d'interfaces dynamiques
- Navigation



# Fonctions

# Les fonctions

## Syntaxe de base

```
func calculerSomme(a: Int, b: Int) -> Int {  
    return a + b  
}
```

```
let resultat = calculerSomme(a: 5, b: 3)  
print("La somme de 5 et 3 est : \resultat")
```

# Les fonctions

## Paramètre par défaut

```
func somme(a: Int = 0, b: Int = 0) -> Int {  
    return a + b  
}
```

```
print(somme()) // renvoie 0  
print(somme(a: 2)) // renvoie 2  
print(somme(a: 2, b: 3)) // renvoie 5
```

# Les fonctions

## Labels d'arguments

- nombre 1 et nombre 2 sont les **labels externes** (utilisé à l'appel)
- a et b sont les **nom interne** (utilisé dans le corps de la fonction)
- Il est aussi possible de masquer le label externe avec `_`

```
func somme(nombre1 a : Int, nombre2 b: Int) -> Int {  
    a + b  
}  
print(somme(nombre1: 5, nombre2: 5))  
|
```

```
func somme(_ a : Int, _ b: Int) -> Int {  
    a + b  
}  
print(somme(5, 5))  
|
```

# Struct

# Struct syntaxe de base

Une struct (structure) est un type personnalisé qui permet de regrouper plusieurs valeurs et comportements (fonctions) liés dans un seul bloc de code.

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello there!")  
    }  
}
```

# Struct :

## Memberwise initializer

- En Swift, les struct génèrent automatiquement un **initialiseur par défaut** basé sur leurs propriétés.
- Le memberwise initializer ne propose que les propriétés qui n'ont pas de valeur par défaut.

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello there!")  
    }  
}  
  
let person = Person(name: "Jasmine") // Memberwise initializer
```

# Struct

## Particularité

- Les fonctions dans une struct ne peuvent pas modifier ses propriétés,

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello, my name is \(name).")  
    }  
}
```

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello, my name is \(name).")  
    }  
  
    func changeName(to newName: String) {  
        name = newName  
    }  
}
```

Cannot assign to property: 'self' is immutable

# Struct

## Particularité

- **Les fonctions dans une struct ne peuvent pas modifier ses propriétés, sauf si elles sont marquées avec le mot-clé mutating.**

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello, my name is \(name).")  
    }  
}
```

```
struct Person {  
    var name: String  
  
    func sayHello() {  
        print("Hello, my name is \(name).")  
    }  
  
    mutating func changeName(to newName: String) {  
        name = newName  
    }  
}
```

# Exercices

# Protocols

# Les protocols

- Un **protocol** définit un **plan (contrat)** composé de méthodes, de propriétés, et d'autres exigences qui décrivent un **comportement spécifique** ou une **fonctionnalité attendue**.
- Lorsqu'un type adopte un protocol, il s'engage à implémenter toutes les méthodes et propriétés requises par ce contrat.

# Les protocols

## Application

- Pour faire en sorte qu'un type (comme une struct) **adopte** un protocol, il suffit d'indiquer le nom du protocole **après le nom du type avec « : »**
- Si un type ne se conforme pas à un protocole, vous obtiendrez une erreur du type : « **does not conform to protocol** »
- La plupart du temps, Xcode vous propose automatiquement de “fixer” l’erreur. En cliquant sur le bouton Fix, il ajoute les propriétés et fonctions requises pour que votre type se conforme au protocole.

```
struct Person : CustomStringConvertible {  
  
    var name: String  
    var age : Int  
  
    func sayHello() {  
        print( "Hello, my name is \(name).")  
    }  
}
```



# Les protocols

## Création

```
protocol FullyNamed {  
    var fullName: String { get }  
  
    func sayFullName()  
}
```

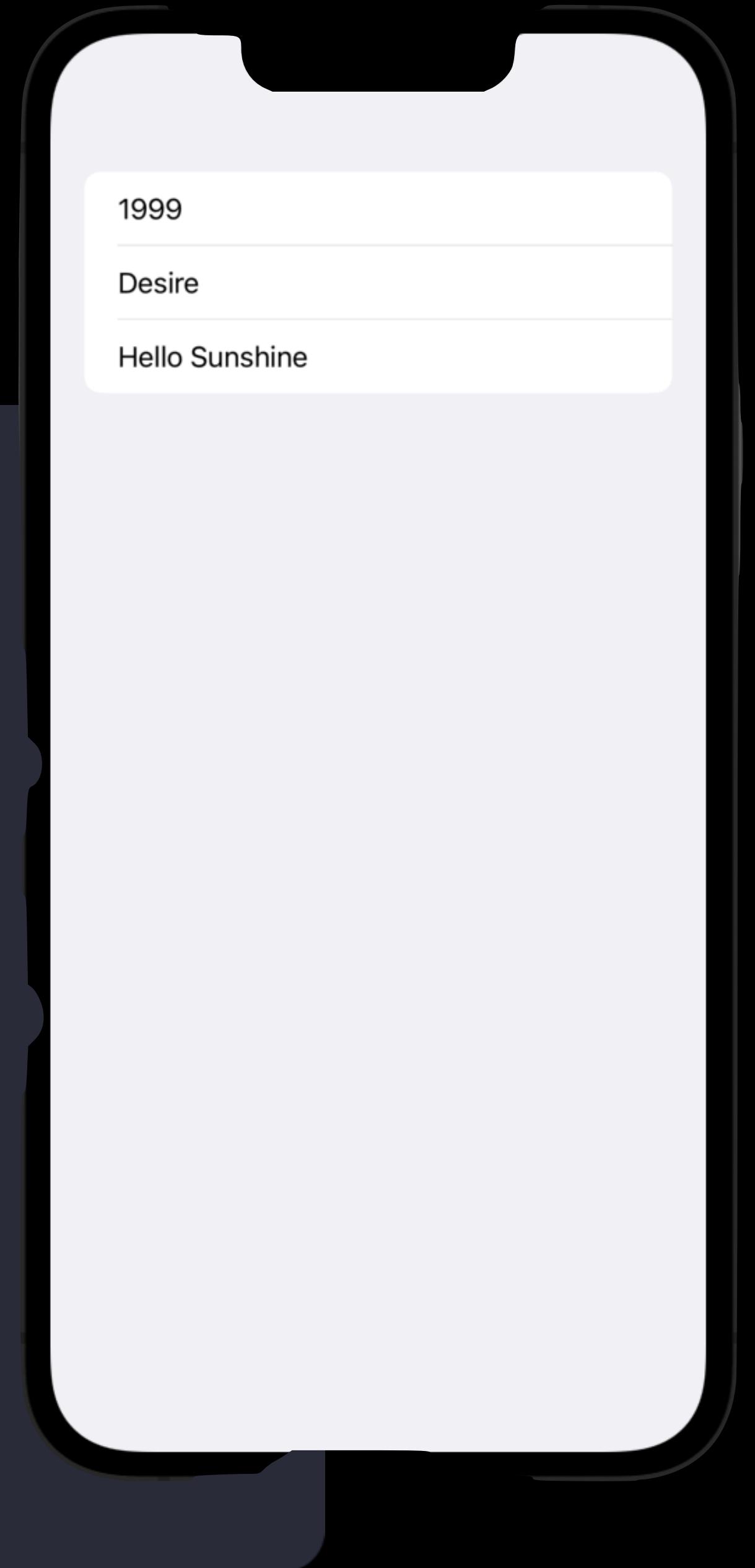
```
struct Person: FullyNamed {  
    var firstName: String  
    var lastName: String  
  
    var fullName: String {  
        return "\(firstName) \(lastName)"  
    }  
  
    func sayFullName() {  
        print(fullName)  
    }  
}
```

# Exercices

# Création d'interface dynamique

# Listes dynamiques

```
struct ContentView: View {  
  
    var albums = [  
        Album(name: "1999"),  
        Album(name: "Desire"),  
        Album(name: "Hello Sunshine")  
    ]  
  
    var body: some View {  
  
        List(albums) { album in  
            Text(album.name)  
        }  
    }  
}
```



# Listes dynamiques

Model de donnée  
Struct

```
struct Album: Identifiable {  
    var id = UUID()  
    var name: String  
}
```

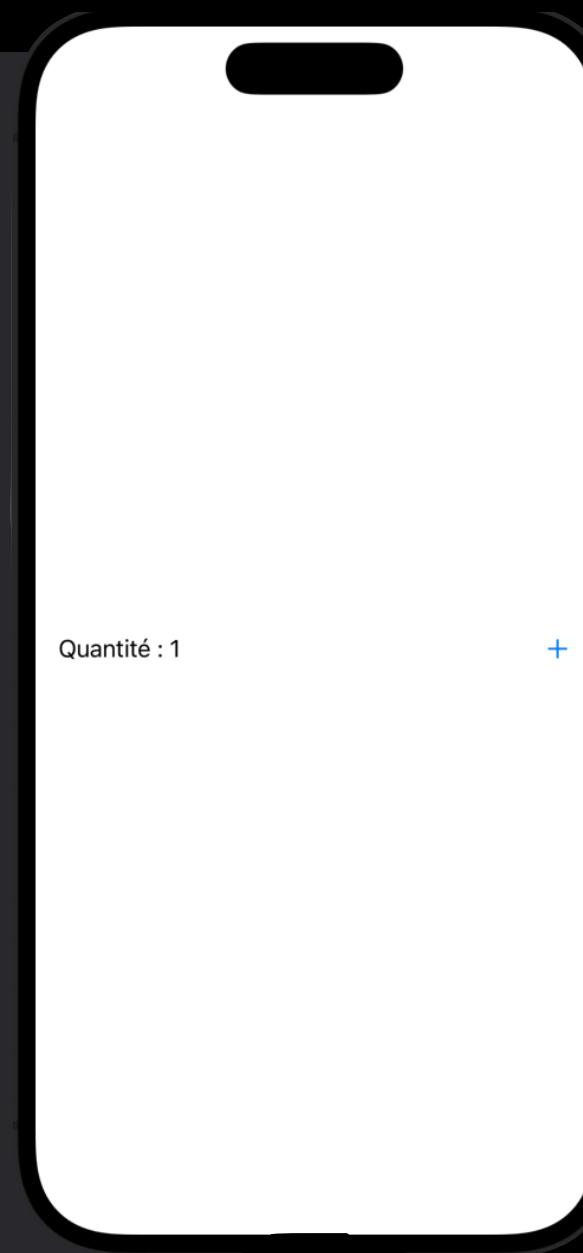
# Interface dynamique

@state

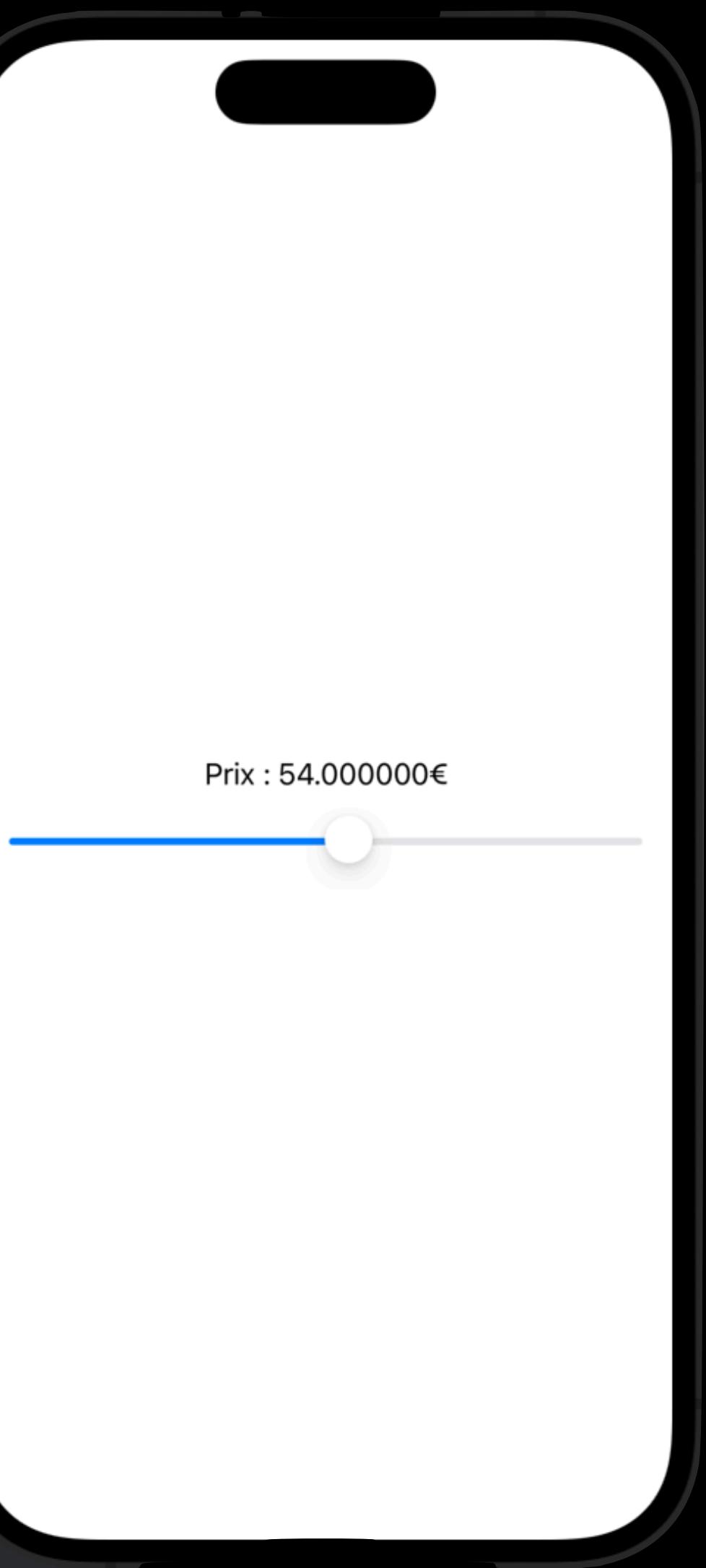
Si des données sont susceptibles d'être modifiées par l'utilisateur pendant l'utilisation de l'application, il faut les rendre **observables** en les déclarant avec `@State`.

```
struct BasketView: View {
    @State var productQuantity = 0

    var body: some View {
        HStack{
            Text("Quantité : \(productQuantity)")
            Spacer()
            Button {
                productQuantity += 1
            } label: {
                Image(systemName: "plus")
            }
        }
        .padding(.horizontal)
    }
}
```



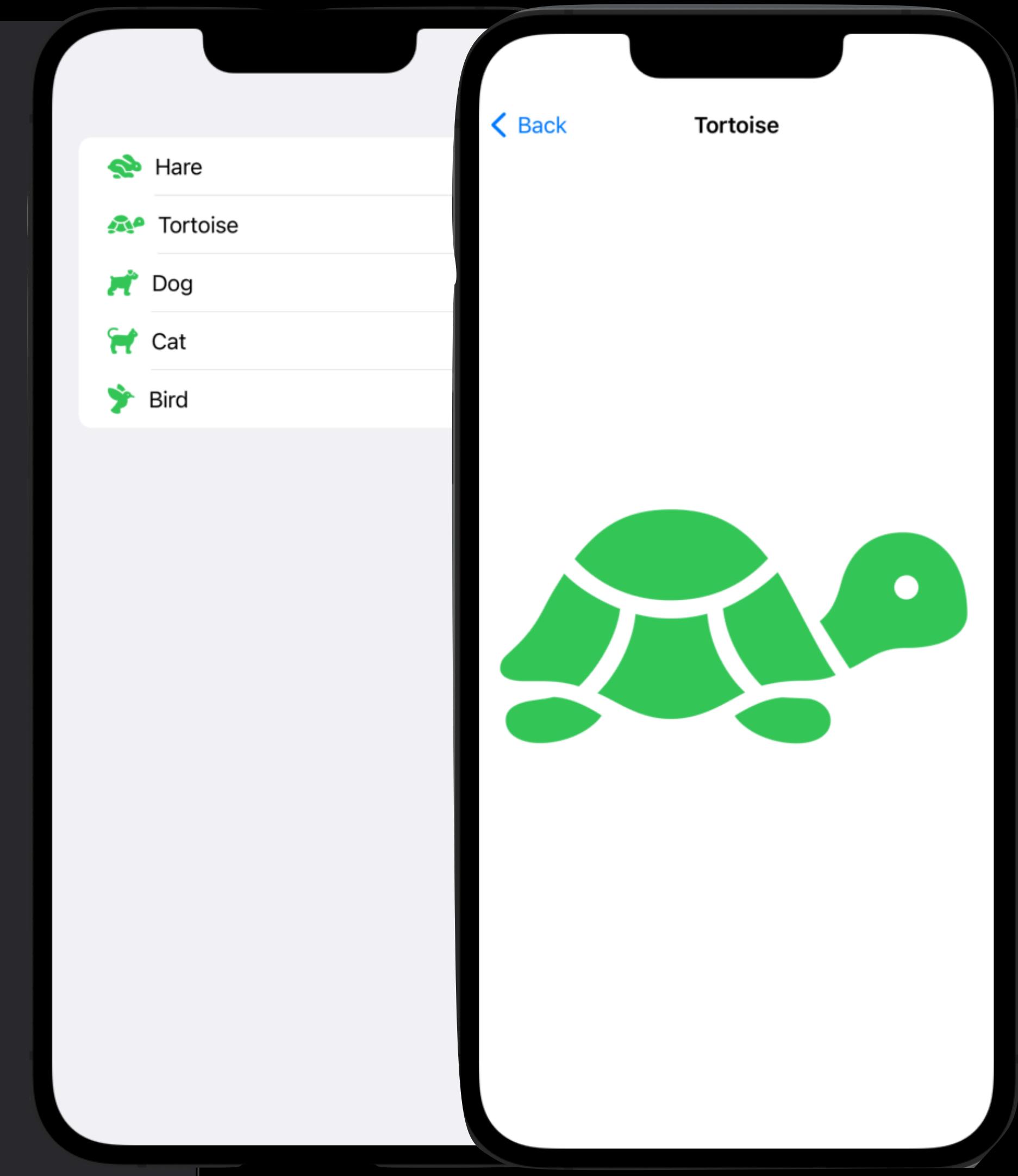
```
struct ProductView: View {  
    @State var productPrice = 0.0  
  
    var body: some View {  
        VStack{  
            Text("Prix : \(productPrice)€")  
            Slider(value: $productPrice, in: 0...100, step: 1)  
        }  
        .padding(.horizontal)  
    }  
}
```



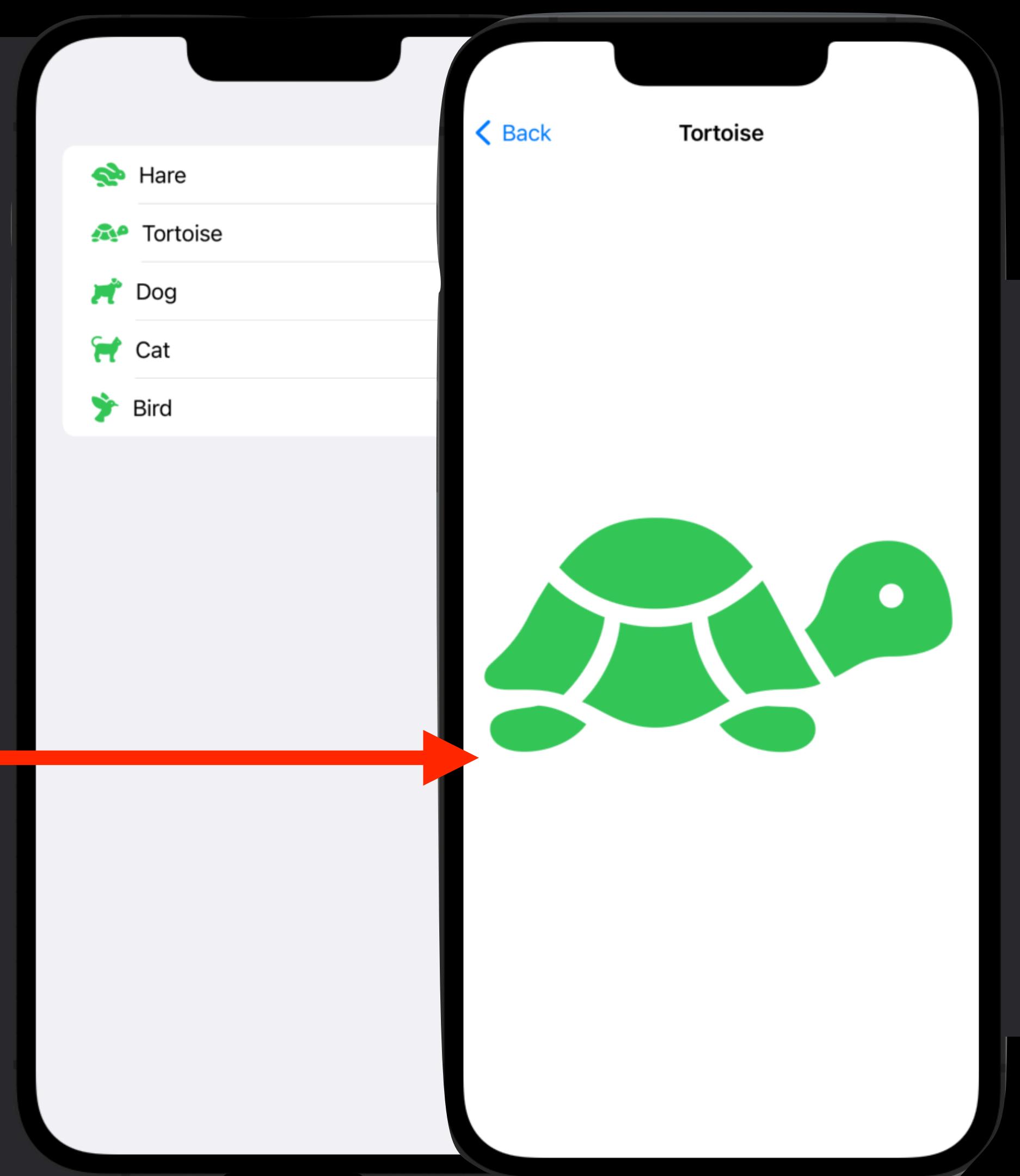
# Navigation

# Navigation

```
struct ListAnimalsView: View {  
    let animals: [Animal] = [  
        Animal(name: "Hare", image: "hare.fill"),  
        Animal(name: "Tortoise", image: "tortoise.fill"),  
        Animal(name: "Dog", image: "dog.fill"),  
        Animal(name: "Cat", image: "cat.fill"),  
        Animal(name: "Bird", image: "bird.fill")  
    ]  
  
    var body: some View {  
        NavigationStack {  
            List(animals) { animal in  
                NavigationLink {  
                    AnimalDetail(data: animal)  
                } label: {  
                    HStack{  
                        Image(systemName: animal.image)  
                            .font(.headline)  
                            .foregroundColor(.green)  
                        Text(animal.name)  
                    }  
                }  
            }  
        }  
    }  
}
```



```
ill"),  
rtoise.fill"),  
l"),  
l"),  
ill")  
  
)  
  
l.image)  
  
reen)
```

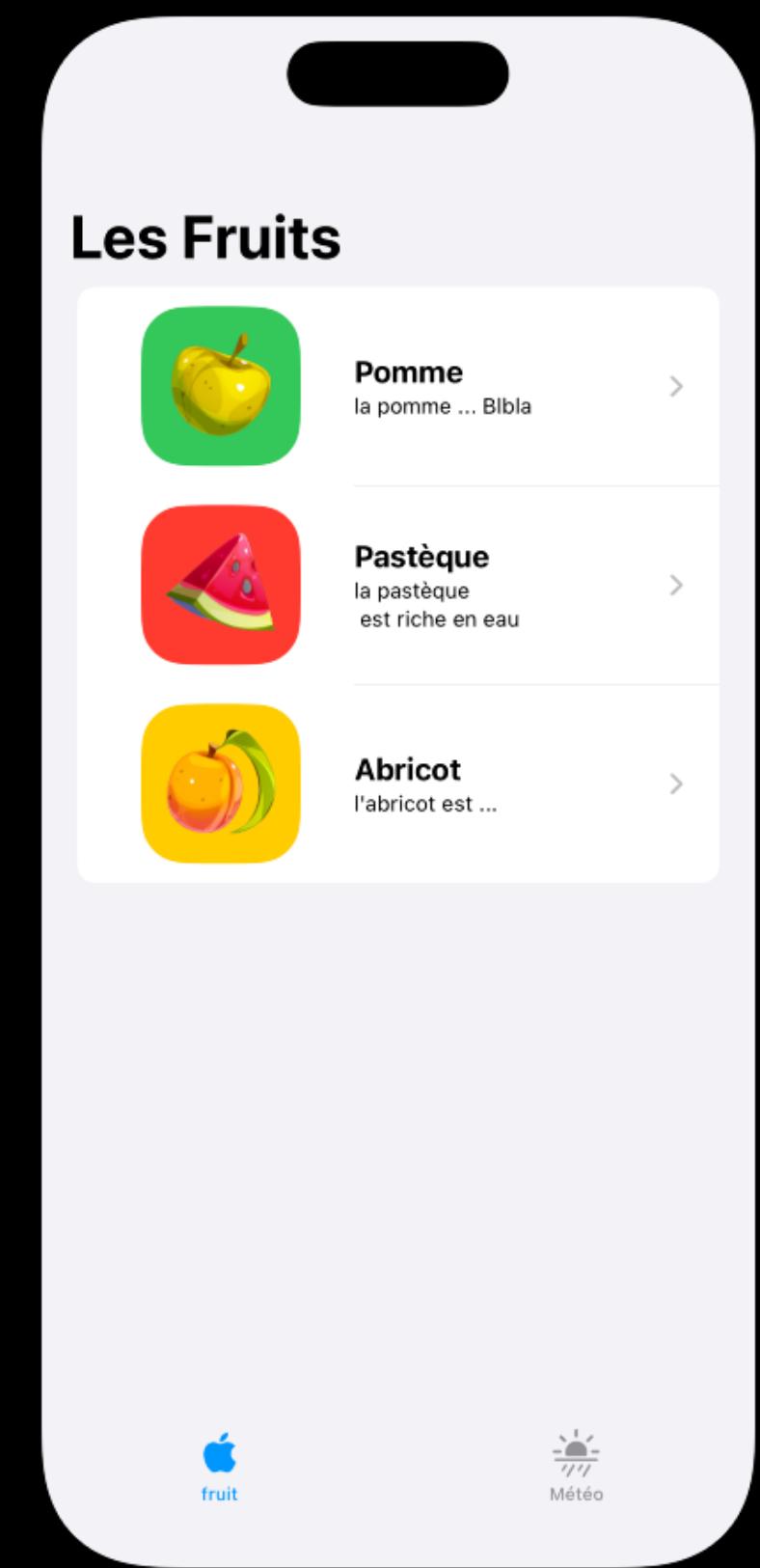


```
struct AnimalDetail: View {  
    var data: Animal  
  
    var body: some View {  
        Image(systemName: data.image)  
            .resizable()  
            .scaledToFit()  
            .padding()  
            .foregroundColor(.green)  
            .navigationTitle(data.name)  
    }  
}
```

# TabView

## Navigation par silo

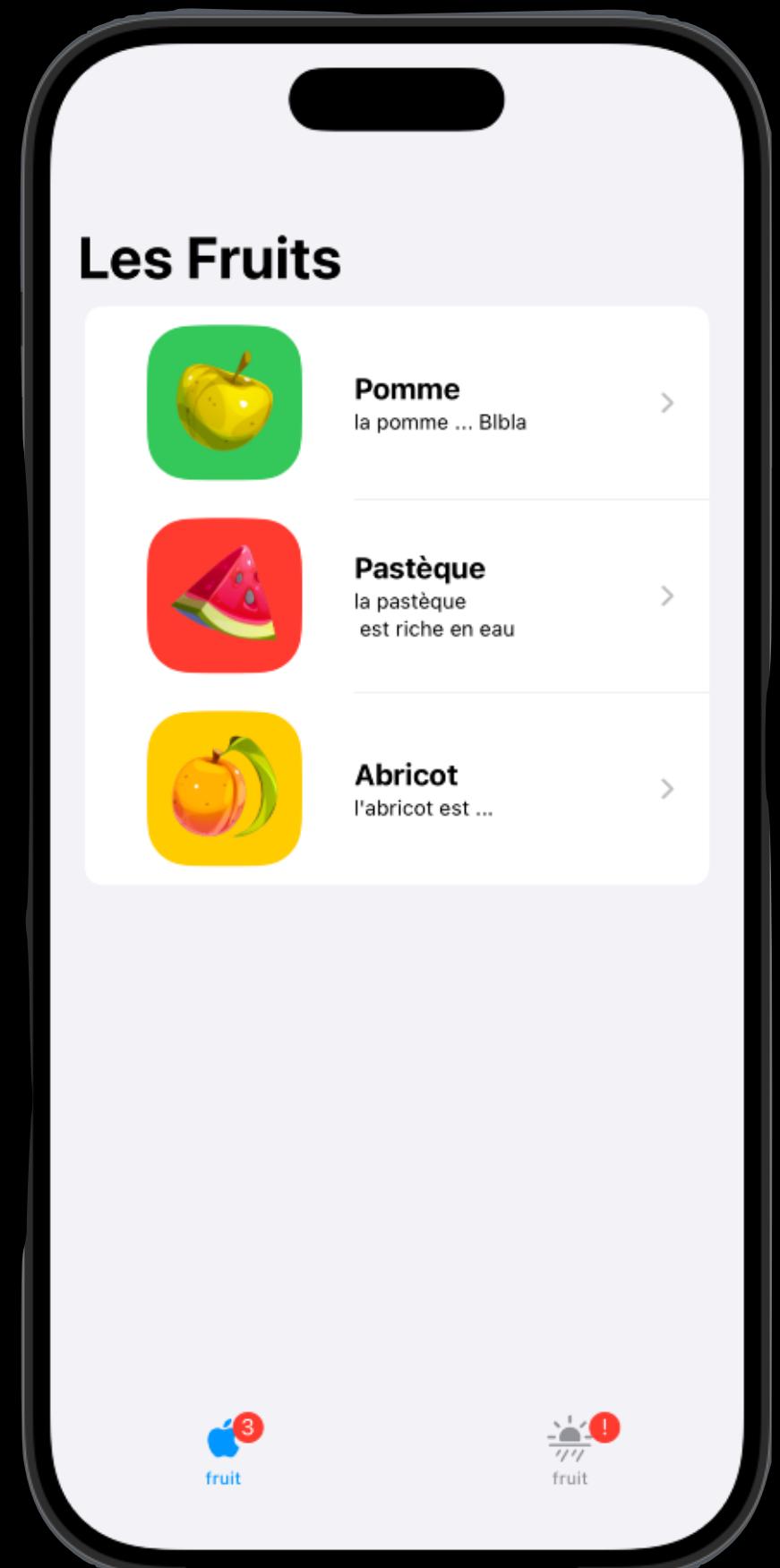
```
struct TabViewTest: View {  
  
    var body: some View {  
  
        TabView {  
  
            FruitListView().tabItem {  
                Label("fruit", systemImage: "apple.logo")}  
            WeatherList().tabItem {  
                Label("Météo", systemImage: "sun.rain")  
            }  
        }  
    }  
}
```



# TabView (depuis iOS 18)

## Navigation par silo

```
var body: some View {  
  
    TabView {  
  
        Tab("fruit", systemImage: "apple.logo") {  
            FruitListView()  
        }  
        .badge(3)  
  
        Tab("fruit", systemImage: "sun.rain") {  
            WeatherList()  
        }.badge("!")  
    }  
}
```



# Transfert et modification de données entre vues

## @Binding

Quand une vue enfant a besoin d'accéder à une donnée déclarée dans le parent, et de la modifier, on utilise une liaison (@Binding).

Le parent donne la référence (\$variable), l'enfant la reçoit sous forme de @Binding.

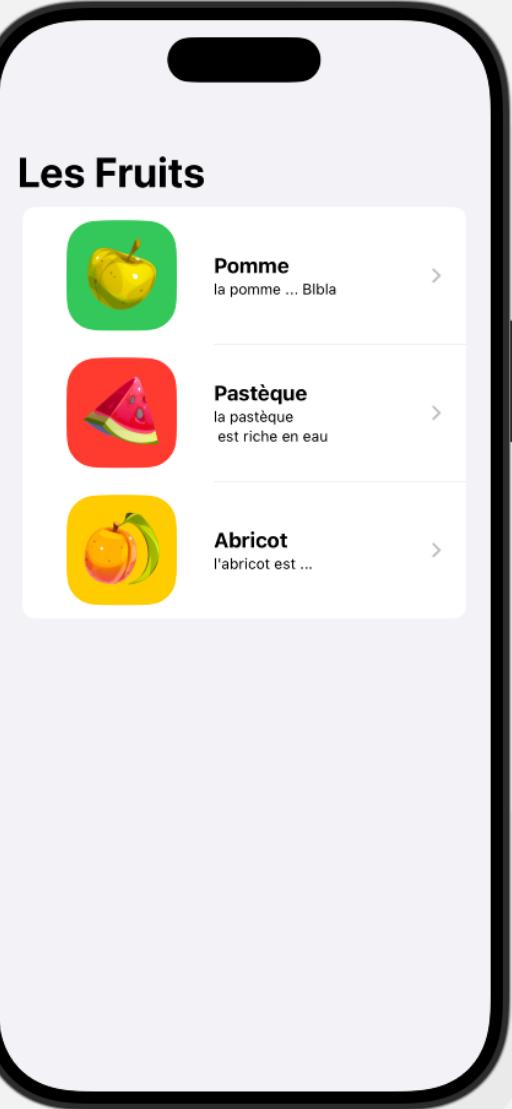
# Transférer des données

## Sans modification

```

6 ///
7
8 import SwiftUI
9
10 struct FruitListView: View {
11
12     let fruits = [
13         Fruit(imageURL: "apple", fruitName: "Pomme", description: "la pomme ... Blbla", bgColor: .green),
14         Fruit(imageURL: "waterMelon", fruitName: "Pastèque", description: "la pastèque \n est riche en eau ", bgColor: .red),
15         Fruit(imageURL: "apricot", fruitName: "Abricot", description: "l'abricot est ... ", bgColor: .yellow )
16     ]
17
18     var body: some View {
19
20         NavigationStack {
21
22             List(fruits) { fruit in
23                 NavigationLink(destination: DetailFruitView(fruit: fruit)) {
24                     FruitDetail(fruit: fruit)
25                 }
26             }
27         }.navigationTitle("Les Fruits")
28
29     }
30
31 }
32
33 }
34
35 }
36
37 }
38
39 }
40
41 #Preview {
42     FruitListView()
43
44 }

```

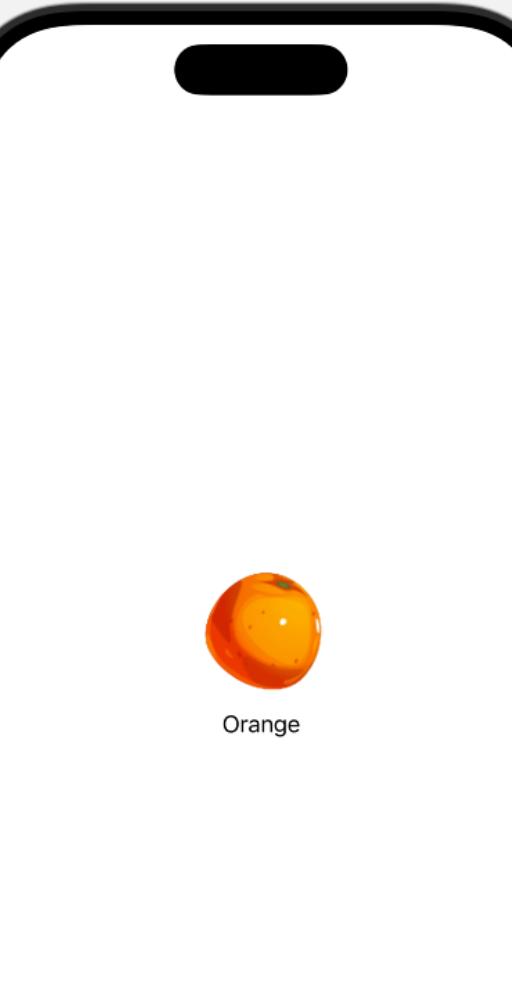


Portrait

```

7
8 import SwiftUI
9
10 struct DetailFruitView: View {
11     let fruit : Fruit
12     var body: some View {
13         VStack {
14             Image(fruit.imageURL).resizable().scaledToFit().frame(height: 100)
15             Text(fruit.fruitName)
16             // TextField("description", text: $fruit.description)
17         }.navigationTitle(fruit.fruitName)
18     }
19 }
20
21
22
23
24
25 #Preview {
26     DetailFruitView(fruit: Fruit(imageURL: "orange", fruitName: "Orange", description: "Banane", bgColor: .black))
27 }
28
29
30
31

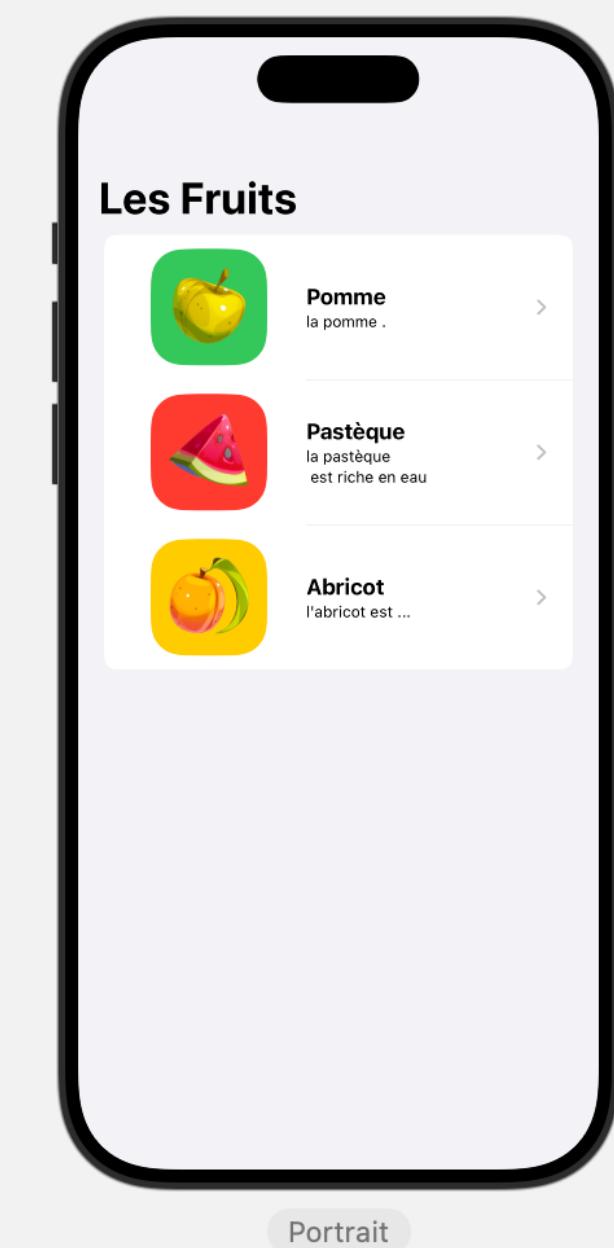
```



# Transférer des données

Avec possibilité de modification

```
8 import SwiftUI
9
10 struct FruitListView: View {
11
12     @State var fruits = [
13         Fruit(imageURL: "apple", fruitName: "Pomme", description: "la pomme ... Blbla", bgColor: .green),
14         Fruit(imageURL: "waterMelon", fruitName: "Pastèque", description: "la pastèque \n est riche en eau ", bgColor:
15             .red),
16         Fruit(imageURL: "apricot", fruitName: "Abricot", description: "l'abricot est ... ", bgColor: .yellow )
17     ]
18
19     var body: some View {
20
21         NavigationStack {
22
23             List($fruits) { fruit in
24                 NavigationLink(destination: DetailFruitView(fruit: fruit)) {
25                     FruitDetail(fruit: fruit)
26                 }
27             }.navigationTitle("Les Fruits")
28
29
30         }
31     }
32 }
33
34
35
36
37 }
38
39 }
```

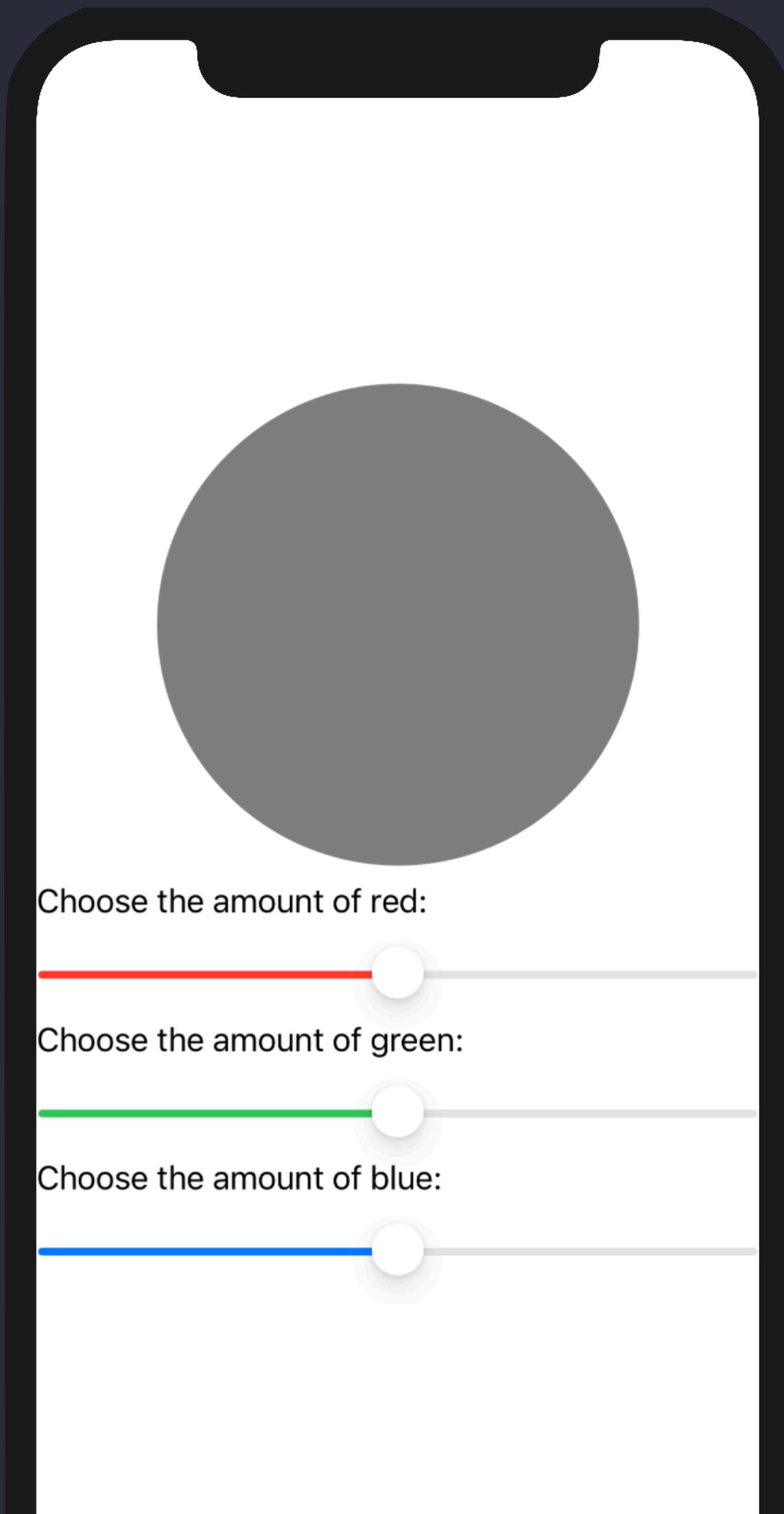


Portrait

```
8 import SwiftUI
9
10 struct DetailFruitView: View {
11     @Binding var fruit : Fruit
12
13     var body: some View {
14         VStack {
15             Image(fruit.imageURL).resizable().scaledToFit().frame(height: 100)
16             Text(fruit.fruitName)
17             TextField("description", text: $fruit.description)
18         }.navigationTitle(fruit.fruitName)
19     }
20
21
22
23
24
25 //##Preview {
26 //    DetailFruitView(fruit: Fruit(imageURL: "orange", fruitName: "Orange", description: "Banane", bgColor: .black))
27 //}
28
29
30 #Preview {
31     DetailFruitView(fruit: .constant(Fruit(imageURL: "banane", fruitName: "Banane", description: "Banane", bgColor: .black)))
32 }
33 }
```

# Extraction et @Binding

```
struct ColorPickerView: View {  
  
    @State var red: Double = 0.5  
    @State var green: Double = 0.5  
    @State var blue: Double = 0.5  
  
    var body: some View {  
        VStack {  
            Color(red: red, green: green, blue: blue, opacity: 1)  
                .frame(width: 250, height: 250)  
                .mask(Circle())  
            VStack(alignment: .leading) {  
                Text("Choose the amount of red:")  
                Slider(value: $red, in: 0...1) {  
                    Text("Red Slider")  
                }  
                .accentColor(.red)  
            }  
            VStack(alignment: .leading) {  
                Text("Choose the amount of green:")  
                Slider(value: $green, in: 0...1) {  
                    Text("Green Slider")  
                }  
                .accentColor(.green)  
            }  
            VStack(alignment: .leading) {  
                Text("Choose the amount of blue:")  
                Slider(value: $blue, in: 0...1) {  
                    Text("Blue Slider")  
                }  
                .accentColor(.blue)  
            }  
        }  
    }  
}
```



## Fichier ColorPicker.swift

```
struct ColorPickerView: View {  
  
    @State var red: Double = 0.5  
    @State var green: Double = 0.5  
    @State var blue: Double = 0.5  
  
    var body: some View {  
        VStack {  
            Color(red: red, green: green, blue: blue, opacity: 1)  
                .frame(width: 250, height: 250)  
                .mask(Circle())  
            ColorPickerSlider(colorValue: $red, colorName: "red", color: .red)  
            ColorPickerSlider(colorValue: $green, colorName: "green", color: .green)  
            ColorPickerSlider(colorValue: $blue, colorName: "blue", color: .blue)  
        }  
    }  
}
```

## Fichier ColorPickerSlider.swift

```
struct ColorPickerSlider: View {  
  
    @Binding var colorValue: Double  
    let colorName: String  
    let color: Color  
  
    var body: some View {  
        VStack(alignment: .leading) {  
            Text("Choose the amount of \(colorName):")  
            Slider(value: $colorValue, in: 0...1) {  
                Text("\(colorName.capitalized) Slider")  
            }  
            .accentColor(color)  
        }  
    }  
}
```