STATES Y BINDINGS

CAMBIAR DINÁMICAMENTE LA APARIENCIA DE UN BOTÓN

Para conocer la acción que se está ejecutando lo primero que tendríamos que hacer al inicio de todo entre la contentView y el body es declarar una variable privada, en este caso booleana y normalment para elegir el nombre siempre empezaría por "is" (en inglés -> ¿es o está? Por defecto la variable la declaramos a false. Luego para cambiar el estado muchas veces usaríamos los ternarios (is… ? __ : __).
Cuando además quremos que SwiftUI se encargue de la actualización automática lo que hacemos es marcar la variable con la directriz "@State", es decir, cuando ocurra cualquier cambio en "is…", SwiftUI calculará automáticamente todas las vistas que hacen referencia a la variable "is…" y las volverá a pintar.
Ahora para que el botón haga la acción de cambiar el estado podríamos hacer **"is…toggle()",** el toggle() siempre cambia la acción de un Booleano de true a false o de false a true. Pero como la acción en el botón se ejecuta dentro de un "closure" y éste es una realidad paralela, si se hace suso de una variable **NO** declarada en el interior del closure, que esa variable esté marcada como: "a que clase pertenece" o "a que estructura pertenece". Como en este caso la variable "is…" pertenece a la misma estructura ContentView, cuando esto sucede, es decir, que necesitamos usar en closure una variable declarada fuera de él pero englobada en la misma clase o estructura donde está el closure, tenemos que ponerle delante "**self**"

Para llamar dentro de un texto en SwiftUI a una variable de tipo Int hay que hacer un casting: **Text("\(variable)")**

BINDGS – Vínculos:

Comentaremos el caso de este ejercicio como ejemplo:
Extraemos el primer botón (de los 3 del ejemplo) en una subview a la que le cambiamos el nombre (este este ejemplo counterView). Entonces nos surge un problema, y es que la variable que teníamos declarada en al View principal ahora ya no nos sirve, porque al haber declarado una subView el **self** ahora hace referencia a la subView y no a la view.
En cualquier caso la dejamos como estaba y copiamos una igual en la subview (ésta no hace falta que sea privada), pero le vamos a marcar en vez de con la propiedada @State lo haremos con @Bindiding
Luego crearíamos en este caso del ejemplo otra variable llamada color para diferenciar los 3 botones de tipo Color.
Entonces en la subview llamamos a las variables en su sitio, lo de contador y la de color y luego en la vista principal al llamar a la subvista nos dice que le faltan 2 parámetros, el del color es como siempre sin problema .color, pero el de la variable **bindeada** sería el nombre de la variable de **View** pero con el símbolo **$** delante.
La vista principal o vista padre contiene a la subvista o vista hijo. La acción de incrementar el contador se lleva a cabo en la subvista, pero tenemos que tener alguna manera de manejar el estado (State) de la variable que se había creado inicialmente en la vista principal con la subvista. Para ello usamos el **@Binding** con la suview y el símbolo **$** al llamar a la variable en el parámetro de la vista principal. En otras palabras, la vista principal invoca a la subvista y le proporciona el estado a través de una variable bindeada.

Ejemplo botón PLAY – STOP:

```
struct ContentView: View {

   @State private var isPlaying = false

   var body: some View {
      Button {
         self.isPlaying.toggle()
      } label: {
         Image(systemName: isPlaying ? "stop.circle.fill" : "play.circle.fill")
            .font(.system(size: 100))
            .foregroundColor(isPlaying ? .red : .blue)
      }
   }
}

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

Ejemplo Contador con botón + y – y la papelera para borrar y poner el contador a 0:

```
struct clicker: View {

  @State private var accountant = 0

   var body: some View {

      VStack{
         Spacer()

      Button {
         //self.accountant = 0
      } label: {
         /* Text("\(accountant)")
          .padding()
          .font(.system(size: 100, weight: .bold, design: .rounded))
          .foregroundColor(Color(red: 129/255, green: 255/255, blue: 3/255))
          .padding(50)
          .background(Color.black)
          .clipShape(Circle())*/  //Hecho por mi...y lo de debajo es como lo hizo Juan Gabriel

            Circle()
            .frame(width:180, height: 180)
            .foregroundColor(.black)
            .overlay(Text("\(accountant)")
                     .font(.system(size: 80, weight: .bold, design: .rounded))
                     .foregroundColor(Color(red: 129/255, green: 255/255, blue: 3/255))
            )
      }
         HStack{

      Button {
         self.accountant -= 1
      } label: {

      }.buttonStyle(plusMineButtonStyle(plusmine: "-"))

      Button {
         self.accountant += 1

      } label: {

      }.buttonStyle(plusMineButtonStyle(plusmine: "+"))
         }.padding()
         Spacer()

      Button {
         self.accountant = 0
      } label: {

         HStack{
         Image(systemName: "trash")
               .foregroundColor(.white)
               .font(.title)
         Text("Borrar Contador")
               .foregroundColor(.white)
               .font(.system(size: 25, weight: .bold, design: .rounded))


         }

      }.buttonStyle(trashButtonStyle())

   }
```

```
    }
}

struct clicker_Previews: PreviewProvider {
    static var previews: some View {
        clicker()
    }
}

struct plusMineButtonStyle: ButtonStyle{

    var plusmine: String

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
        Rectangle()
            .frame(width: 150, height: 100)
            .foregroundColor(Color(red: 129/255, green: 255/255, blue: 3/255))
            .overlay(Text(plusmine))
            .font(.system(size: 50, weight: .black, design: .rounded))
            .foregroundColor(.black)
            .cornerRadius(50)
            .scaleEffect(configuration.isPressed ? 0.8 : 1.0)
    }
}

struct trashButtonStyle: ButtonStyle{
    func makeBody(configuration: Configuration) -> some View {
        configuration.label
        .padding(.maximum(0, 15))
        .background(.red)
        .cornerRadius(50)
        .scaleEffect(configuration.isPressed ? 0.7 : 1.0)
    }
}
```

Ejemplo de suma de contador con tres botones de distinto color y uno de papelera para poner el contador a cero:

```
struct Vinculo: View {

    @State private var contador = 0
    //@State private var contadorParcial = 0
    @State private var black = 0
    @State private var red = 0
    @State private var blue = 0

    var body: some View {
        VStack{
            Text("\(black + red + blue)")
                .font(.system(size: 100, weight: .bold, design: .rounded))

            CounterView(contador: $black, colorButton: .black)
            CounterView(contador: $red, colorButton: .red)
            CounterView(contador: $blue, colorButton: .blue)

            Papelera(contador: $contador, black: $black, red: $red, blue: $blue)

        }
    }
}

struct binding_Previews: PreviewProvider {
    static var previews: some View {
```

```
        Vinculo()
    }
}

struct CounterView: View {

    @Binding var contador: Int
    //@State var contadorParcial = 0

    var colorButton: Color

    var body: some View {
        Button {
            //self.contadorParcial += 1
            self.contador += 1
        } label: {
            /* Text("\(accountant)")
           .padding()
           .font(.system(size: 100, weight: .bold, design: .rounded))
           .foregroundColor(Color(red: 129/255, green: 255/255, blue: 3/255))
           .padding(50)
           .background(Color.black)
           .clipShape(Circle())*/  //Hecho por mi...y lo de debajo es como lo hizo Juan Gabriel

            Circle()
                .frame(width:160, height: 160)
                .foregroundColor(colorButton)
                .overlay(Text("\(contador)")
                            .font(.system(size: 80, weight: .bold, design: .rounded))
                            .foregroundColor(Color(red: 129/255, green: 255/255, blue: 3/255))
                )
        }
    }
}

struct Papelera: View {

    @Binding var contador: Int
    @Binding  var black : Int
    @Binding  var red : Int
    @Binding  var blue : Int

    var body: some View {
        Button {
            contador = 0
            black = 0
            red = 0
            blue = 0


        } label: {
            Image(systemName: "trash")
                .frame(minWidth:0, maxWidth: .infinity)
                .padding()
                .font(.largeTitle)
                .background(.red)
                .foregroundColor(.white)
                .cornerRadius(50)
                .padding()
        }
    }
}
```