# Desarrollo de Apps para iOS
# SwiftUI

IWEB 2019-2020
Santiago Pavón

ver: 2019.12.09

© Santiago Pavón - UPM-DIT

# Introducción

- Introducido con iOS 13 y Xcode 11.
- Desarrollo del UI de forma declarativa.
    - Declarar como es el UI.
    - No se implementa el UI proporcionando UIViews, no se usa Storyboard, no hay controladores, no hay autolayout
- Crear un proyecto: Seleccionar que el User Interface es SwiftUI.
- El canvas muestra un Preview del UI.
    - Editar en el código o en el canvas.
    - Actualizar el canvas: pulsar en Resume, vista en vivo o en un dispositivo real.
    - Command+Click sobre un elemento para ver el Inspector,
    - Usar el inspector de atributos: seleccionar vistas y modificadores
-

© Santiago Pavón - UPM-DIT

# Repaso de Swift

- Inferencia de tipos.
- Tipos opacos: **some View**
- Struct es un tipo valor.
- Memberwise Initializator: Constructor de un Struct que asigna valores a las propiedades sin inicializar.

```
struct Persona {
   var nombre
   var edad
}
var p = Persona(nombre: "Ana", edad: 19)
```

- Trailing Closures: Si el último parámetro de una función es una closure, al invocar la función puede ponerse la expresión closure fuera de los paréntesis.

© Santiago Pavón - UPM-DIT

# Demo 1

- Crear un proyecto son SwiftUI como tipo de interface.
- La vista creada es ContentView.
  - La propiedad body devuelve la vista a crear.
  - Solo devuelve una view.
  - Crear varias views y usar VStack para agruparlas.
- La clase ContentView_Previews implementa la(s) preview(s).
  - En la propiedad previews.
  - Cambiar por lo que se quiera.
  - Modificadores:

```
.previewLayout(.fixed(width: 100, height: 100))

.previewDevice(PreviewDevice(rawValue: "iPhone SE"))

.previewDisplayName(deviceName)
```

- El contenido de la UIWindow de la app se crea en SceneDelegate.
  - Cambiar en SceneDelegate la vista ContentView por la view que se quiera usar inicialmente.

© Santiago Pavón - UPM-DIT

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, World!")
            Text("Adios")
            Button(action: {
                print("Hola")
            }) {
                Text("Demo")
            }
        }
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ForEach(["iPhone SE", "iPhone XS Max"], id: \.self) { deviceName in
            ContentView()
                .previewLayout(.fixed(width: 250, height: 400))
                .previewDevice(PreviewDevice(rawValue: deviceName))
                .previewDisplayName(deviceName)
        }
    }
}
```

Mostrar el canvas en el editor y pulsar el botón Resume

© Santiago Pavón - UPM-DIT

# Views

- Las views de SwiftUI son:
  - Text, Image
  - TextField, SecureField
  - Button, SegmentedControl, Stepper, Toggle, Slider
  - Picker, DatePicker
  - Spacer, Divider
  - VStack, HStack, ZStack, ScrollView
  - List, Form
  - NavigationView, TabView,
  - Alert, ActionSheet
  -

© Santiago Pavón - UPM-DIT

# Modificadores

- Modificador: función que toma una view y devuelve otra view.
- De la clase base se heredan muchos modificadores comunes, y cada tipo de view también tiene sus propios modificadores.
- Ejemplos:
  - `.background(Color.blue), .foregroundColor(.red)`
  - `.font(.title) .fontWeight(.black)`
  - `.padding(),.padding(.bottom, 100)`
  - `.overlay(Circle().stroke(Color.gray, lineWidth: 2))`
  - `.clipShape(Circle())`
  - `.shadow(radius: 10)`
  - `.frame(width: 100),.frame(width: 100, height: 100)`
  - `.offset(x:10, y: -50)`
  - `.edgesIgnoringSafeArea(.top)`
  - `.rotationEffect(0.8, anchor: .bottom)`
  - `.onAppear { ... },.onDisappear { ... }`
  - `.disabled(true)`
- El orden de aplicación de los modificadores importa.

© Santiago Pavón - UPM-DIT

- Crear modificadores personalizados:
  - Hay que definir un struct conforme con el protocolo **ViewModifier**.
  - Este protocolo requiere que se implemente el método **body(content: Content)**.
    - Hay que usar el parámetro **content** para aplicar nuestras transformaciones.

```
struct MiEstiloText: ViewModifier{
    func body(content: Content) -> some View {
        content
            .foregroundColor(.yellow)
            .background(Color.blue)
            .font(.title)
    }
}

struct EjemploView: View {
    var body: some View {
        Text("Hola Mundo")
            .foregroundColor(.green)
            .modifier(MiEstiloText())
    }
}
```

© Santiago Pavón - UPM-DIT

# Text

```
Text("Hello, World!")
    .background(Color.blue)
    .padding()
```

```
Text("Hello, World!")
    .padding()
    .background(Color.blue)
```

```
Text("Hello,")
    .foregroundColor(.red)
+ Text(" World!")
    .foregroundColor(.blue)
    .fontWeight(.bold)
+ Text("Hola,")
    .foregroundColor(.orange)
    .font(.title)
+ Text(" Mundo!")
    .foregroundColor(.green)
```

Hello, **World!**Hola, Mundo!

© Santiago Pavón - UPM-DIT

# Image

```
Image("cara")
    .resizable()
    .scaledToFit()
    .background(Color.blue)
    .clipShape(Circle())
    .rotationEffect(Angle(degrees: 45))

Image(systemName: "square.and.arrow.up")
    .resizable()
```

Descargar SF Symbols: **`https://developer.apple.com/design/downloads/SF-Symbols.dmg`**

© Santiago Pavón - UPM-DIT

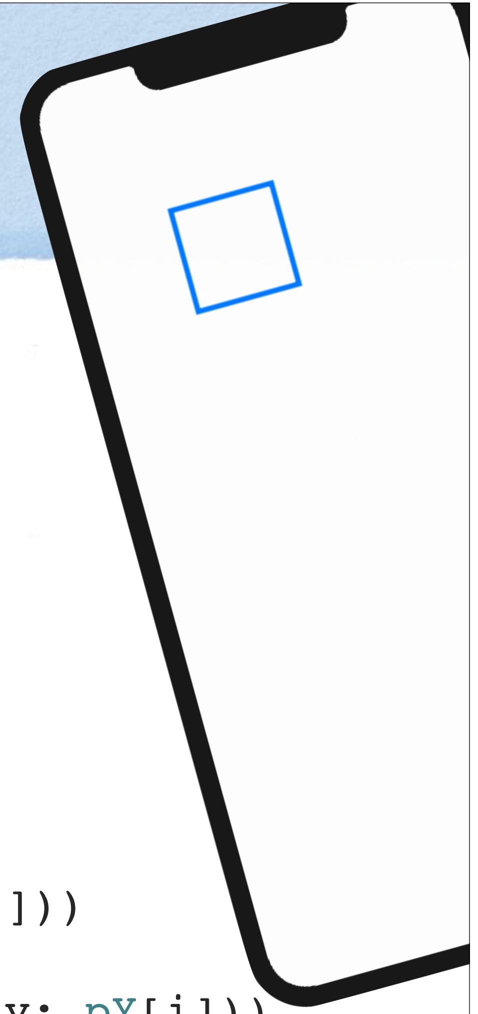
# Button, Toggle

```
Button(action: {
    // Sentencias a ejecutar al pulsar el boton
    print("pulsado")
}) {
    // Contenido del boton
    Image(systemName: "square.and.arrow.up")
        .resizable()
        .frame(width:50, height: 50)
    Text("Subir")
        .font(.title)
}
```

```
Toggle(isOn: $show) {
    Text("Visible")
}
```

© Santiago Pavón - UPM-DIT

# Path (*Dibujar*)



```
import SwiftUI

struct ContentView: View {

    let pX = [ 100, 100, 200, 200]
    let pY = [ 100, 200, 200, 100]

    var body: some View {
        Path { path in
            path.move(to: CGPoint(x: pX[0], y: pY[0]))
            for i in 1...3 {
                path.addLine(to: CGPoint(x: pX[i], y: pY[i]))
            }
            path.closeSubpath()
        }.stroke(Color.blue, lineWidth: 5)
    }
}
```

© Santiago Pavón - UPM-DIT

- Más:

```
path.move(to: CGPoint)
path.addLine(to: CGPoint)
path.addLines([CGPoint])
path.addQuadCurve(to: CGPoint, control: CGPoint)
```

- Usar Fill y Stroke con un Path:

```
var path = Path(CGRect(x: 0, y: 0, width: 100, height:
100))
path.fill(LinearGradient(
                 gradient: Gradient(colors: [.red, .yellow]),
                 startPoint: UnitPoint(x: 0.3, y: 0.3),
                 endPoint: UnitPoint(x: 0.7, y: 0.7)))
    .overlay(path.stroke(Color.blue, lineWidth: 5))
    .frame(width:100, height: 100)
```

© Santiago Pavón - UPM-DIT

# Shape

- Shapes predefinidas:

```
Rectangle()
     .fill(Color.yellow)
     .frame(width: 200, height: 200)

RoundedRectangle(cornerRadius: 35, style: .continuous)
     .fill(Color.red)
     .frame(width: 200, height: 200)

Capsule()
     .fill(Color.green)
     .frame(width: 100, height: 50)

Ellipse()
     .fill(Color.blue)
     .frame(width: 100,height: 50)

Circle()
     .fill(Color.pink)
     .frame(width: 100,height: 50)
```

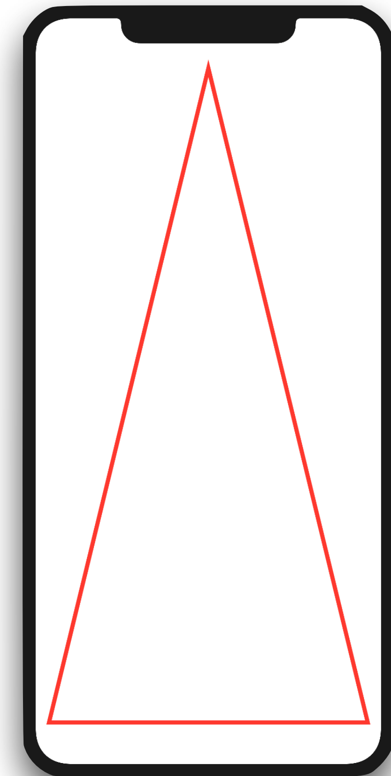
© Santiago Pavón - UPM-DIT

- Shape personalizadas:
  - Para crear nuestras propios paths.
  - Crear un struct que sea conforme con el protocolo Shape.
    - Requiere que se implemente el método:

```
func path(in rect: CGRect) -> Path
```

  - Ejemplo:

```
struct Piramide: Shape {
    func path(in rect: CGRect) -> Path {

        let w = rect.size.width
        let h = rect.size.height

        var path = Path()

        path.move(to: CGPoint(x: 0, y: h))
        path.addLine(to: CGPoint(x: w, y: h))
        path.addLine(to: CGPoint(x: w/2, y: 0))
        path.closeSubpath()

        return path
    }
}
```

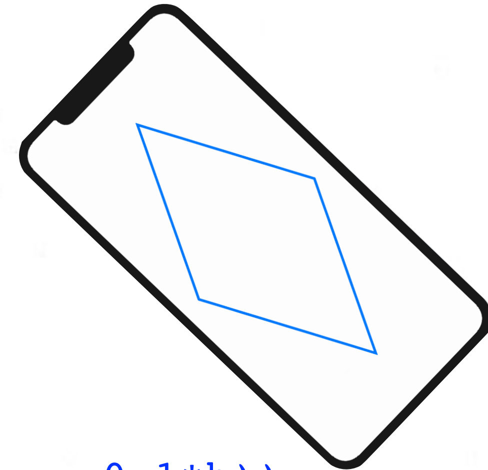


```
Piramide()
.stroke(Color.red, lineWidth: 5)
```

© Santiago Pavón - UPM-DIT

# GeometryReader

- Wrapper de las views que permite acceder a la geometría de la view padre.



```
struct ContentView: View {

    var body: some View {
        GeometryReader { geometry in
            Path { path in
                let w = geometry.size.width
                let h = geometry.size.height

                path.move(to: CGPoint(x: 0.5*w, y: 0.1*h))
                path.addLine(to: CGPoint(x: 0.1*w, y: 0.5*h))
                path.addLine(to: CGPoint(x: 0.5*w, y: 0.9*h))
                path.addLine(to: CGPoint(x: 0.9*w, y: 0.5*h))

                path.closeSubpath()
            }
            .stroke(Color.blue, lineWidth: 5)
        }
    }
}
```

© Santiago Pavón - UPM-DIT

```swift
struct ContentView: View {

    var body: some View {
        GeometryReader {geometry in
            VStack {
                Image("dharma")
                    .resizable()
                    .scaledToFit()
                    .frame(height: geometry.size.height/6)
                Image("dharma")
                    .resizable()
                    .scaledToFit()
                    .frame(height: geometry.size.height/3)
                Image("dharma")
                    .resizable()
                    .scaledToFit()
                    .frame(height: geometry.size.height/2)
            }
        }
    }
}
```

© Santiago Pavón - UPM-DIT

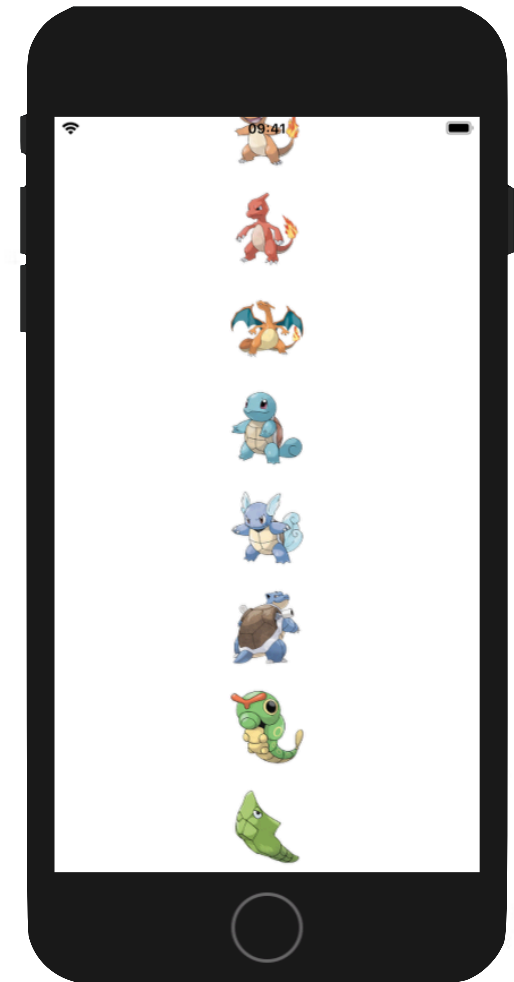
# VStack, HStack, ZStack

```
VStack {
    view
    view
    view
}
VStack(aligment: .leading) {
    views
}
HStack {
    views
}
ZStack(aligment: .bottomtrailing) {
    views
}
```

© Santiago Pavón - UPM-DIT

# ScrollView

```swift
let names = ["001", "002", "003", "004", "005",
       "006", "007", "008", "009", "010", "011"]

var body: some View {
    ScrollView(.vertical, showsIndicators: true) {
        ForEach(names, id: \.self) {name in
            Image(name)}
    }
}
```




© Santiago Pavón - UPM-DIT

# @State

- @State es un Property Wrapper que indica que una propiedad contiene parte del estado de la app.

```
@State var edad: Int = 0
```

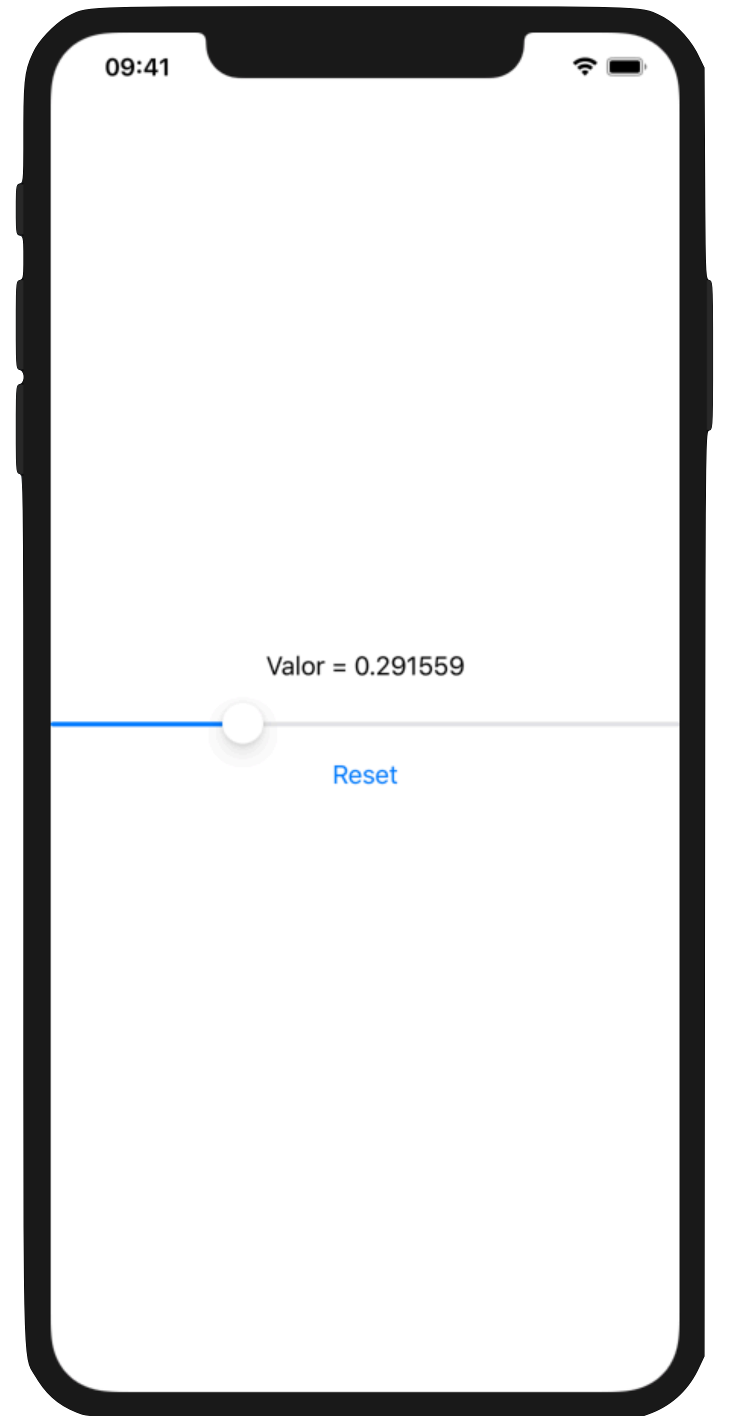
- Si cambia el valor de la propiedad, entonces se refresca la UI.

- Los controles usan un binding a una propiedad de estado para actualizar su valor.
  - Usar el prefijo $ para crear un binding a la propiedad.

- Las propiedades de estado afectan al comportamiento de las views, a su contenido, su layout, ...

© Santiago Pavón - UPM-DIT

```
struct ContentView: View {

    @State var x = 0.5

    var body: some View {
       VStack {
            Text("Valor = \(x)")

            Slider(value: $x)

            Button(action: {
                self.x = 0
            }) {
                Text("Reset")
            }
       }
    }
}
```




© Santiago Pavón - UPM-DIT

# TextField

```
@State var name: String = "pepe"

var body: some View {
    VStack {
        HStack {
            Text("Nombre =")
            TextField("Nombre", text: $name)
                .background(Color.white)

        }
        .padding()
        .background(Color.gray)

        Text("Ha escrito \(name)")
            .font(.largeTitle)
    }
}
```

© Santiago Pavón - UPM-DIT

# Picker, DatePicker

© Santiago Pavón - UPM-DIT

# Shape

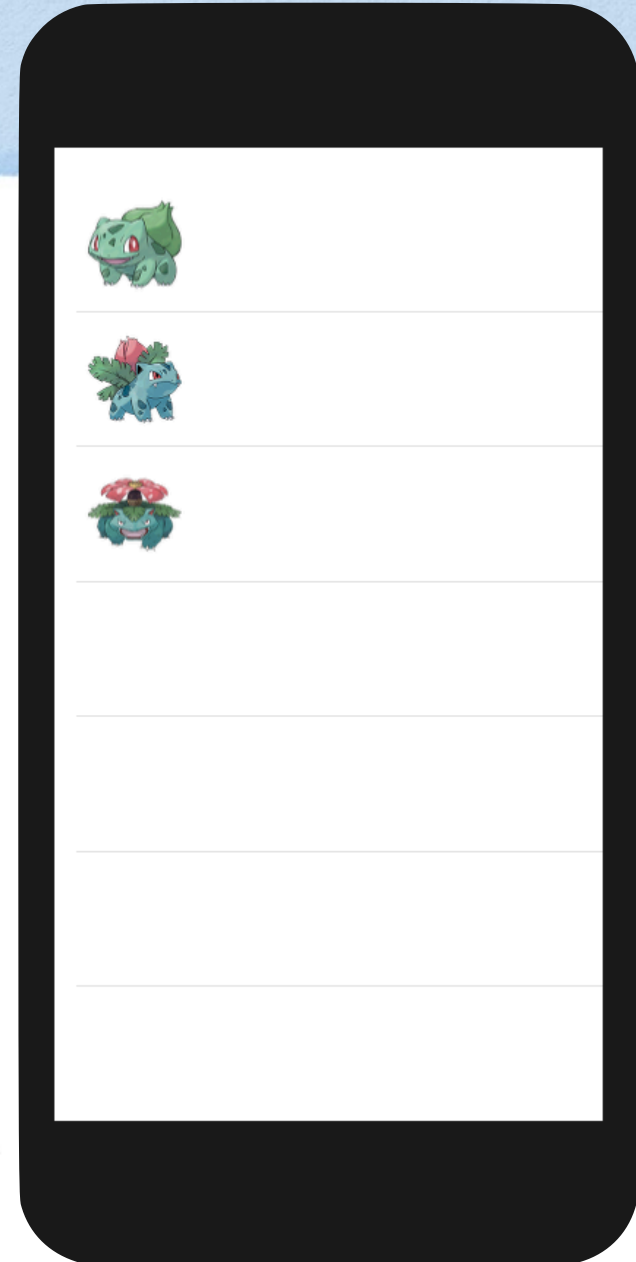
- Protocolo para crear un Path dentro de un rectángulo.

© Santiago Pavón - UPM-DIT

# List

- List es una lista de filas.
- Crearla pasando a List como parámetro una closure con las views.

```
struct ContentView: View {
    var body: some View {
        List {
            Image("001")
            Image("002")
            Image("003")
        }
    }
}
```



© Santiago Pavón - UPM-DIT

- Crear la lista pasando tres argumentos a List:
  - una colección de datos.
  - id qu es un KeyPath que identifica unívocamente cada dato de la colección.
  - una closure que devuelve una view para cada dato de la colección.

```
struct ContentView: View {

    var nombres = ["001", "002", "003"]

    var body: some View {
        List(nombres, id: \.self) { name in
            Image(name)
        }
    }
}
```

© Santiago Pavón - UPM-DIT

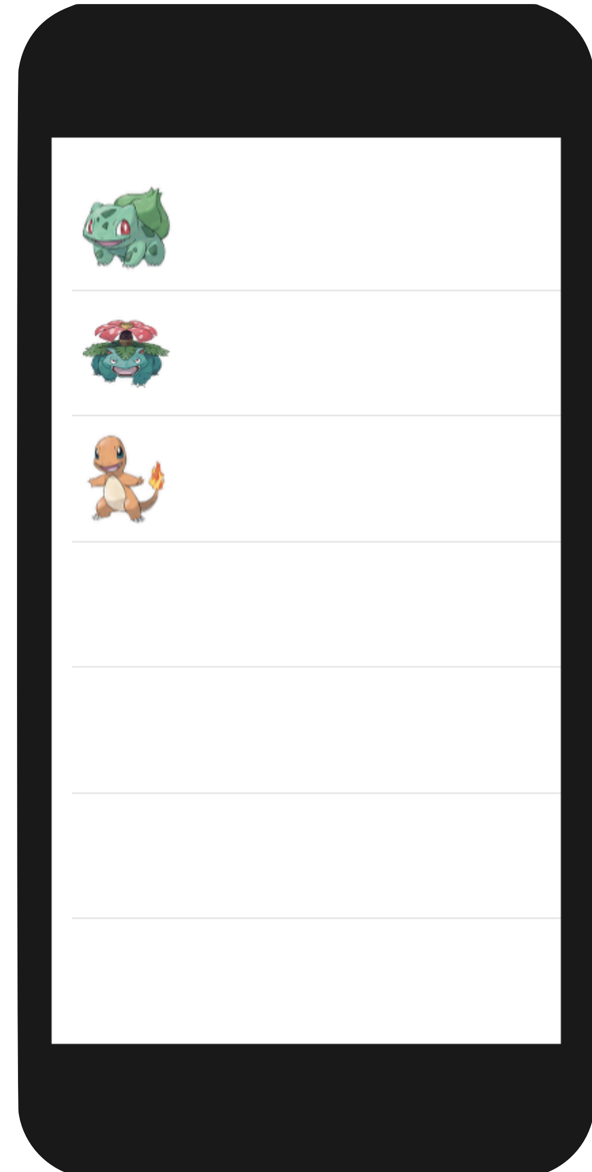
- Crear la lista pasando dos argumentos a List:
  - Si los elementos de la colección de datos son conformes con el protocolo **Identifiable**, puede omitirse el segundo argumento.
  - El protocolo **Identifiable** requiere que el tipo tenga una propiedad **id** que identifique unívocamente a cada instancia.

```
struct Dato: Identifiable {
    var nombre: String
    var id: UUID
}

let datos = [
    Dato(nombre: "001" , id: UUID()),
    Dato(nombre: "002" , id: UUID()),
    Dato(nombre: "003" , id: UUID()),
]

struct ContentView: View {
    var body: some View {
        List(datos) { dato in
            Image(dato.nombre)
        }
    }
}
```

© Santiago Pavón - UPM-DIT

- ForEach
  - Se usa dentro de una List cuando las views a mostrar se eligen dinámicamente. o combinando diferentes tipos de views.
  - Ejemplo: filtrar algunos elementos de array.

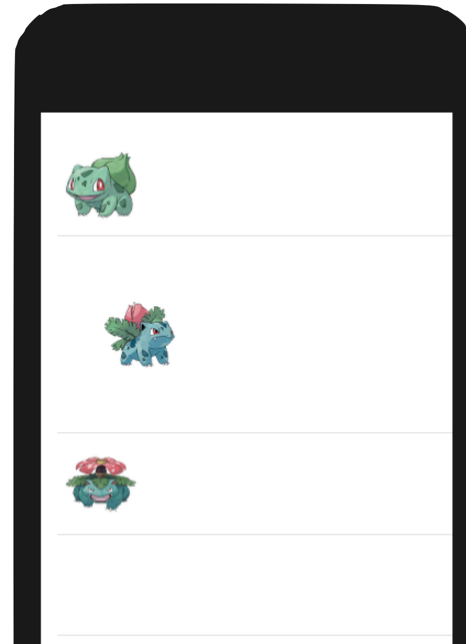
```
struct Dato: Identifiable {
    var nombre: String
    var id: UUID
}

let datos = [
    Dato(nombre: "001" , id: UUID()),
    Dato(nombre: "002" , id: UUID()),
    Dato(nombre: "003" , id: UUID()),
    Dato(nombre: "004" , id: UUID()),
]

struct ContentView: View {

    var body: some View {
        List{
            ForEach(datos) { dato in
                if dato.nombre != "002" {
                    Image(dato.nombre)
                }
            }
        }
    }
}
```



© Santiago Pavón - UPM-DIT

- Insets
  - Puede asignarse un Insets a cada una de las views de la List.

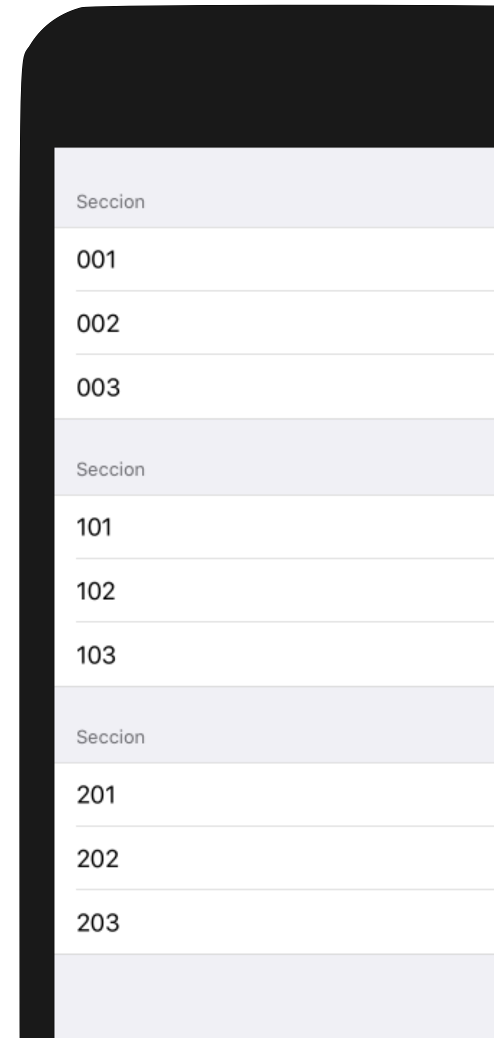
```
struct ContentView: View {
    var body: some View {
        List {
            Image("001")
            Image("002")
                .listRowInsets(EdgeInsets(top: 50,
                                          leading: 50,
                                          bottom: 50,
                                          trailing: 50))
            Image("003")
        }
    }
}
```



© Santiago Pavón - UPM-DIT

- Secciones
  - Usar Section para crear secciones.
  - Usar el modificador **`.listStyle(GroupedListStyle())`** para que el estilo de la lista sea agrupada.

```
var datos = [
    ["001", "002", "003"],
    ["101", "102", "103"],
    ["201", "202", "203"]
]

var body: some View {
    List {
     ForEach(datos, id: \.self) {fila in
             Section(header: Text("Seccion")) {
                 ForEach(fila, id: \.self) { item in
                    Text(item)
                 }
             }
         }
     }
     .listStyle(GroupedListStyle())
}
```




© Santiago Pavón - UPM-DIT

# NavigationView

- Navegación entre pantallas.
- Pasos:
  - Meter una List dentro de un NavigationView.
  - Añadir a la List el siguiente modificador para poner el título:

```
.navigationBarTitle(Text("Demo"))
```

  - Si se quiere que el titulo tenga letra pequeña:

```
.navigationBarTitle(Text("Demo"), displayMode: .inline)
```

  - Para poner views en la barra de navegación se añade el modificador a la List:

```
.navigationBarItems(trailing: unaView)
```

© Santiago Pavón - UPM-DIT

- Meter algunas (o todas) las filas de la List en un NavigationLink:

```
NavigationLink(destination: laViewDestino()) {
    Views_de_la_fila
}
```

  - Notas:
    - En un Navigation Link los textos se pintan con el color accent definido.
      - Para cambiarlo usar el modificador `.foregroundColor(.primary)`.
    - y las imágenes se pintan como una máscara.
      - Para pintarlas tal y como son originalmente, añadir el modificador `.renderingMode(.original)`.

- Añadir items a la barra de navegación:

```
.navigationBarItems(trailing: una_view)
```

© Santiago Pavón - UPM-DIT

```
struct ContentView: View {

    var pokedexModel = PokedexModel()

    var body: some View {
        NavigationView {
            List {
                ForEach(pokedexModel.types, id: \.name) {type in
                    Section(header: HStack {
                        Image(type.icon)
                            .resizable()
                            .frame(width: 50, height: 50)
                        Text(type.name)
                            .font(.largeTitle)
                    }) {
                        ForEach(type.races, id: \.code) { race in
                            NavigationLink(destination: RaceDetail(race: race)) {
                                ItemRace(race: race)
                            }
                        }
                    }
                }
            }
            .listStyle(GroupedListStyle())
            .navigationBarTitle("Pokedex")
        }
    }
}
```

**Ver la demo Pokedex**

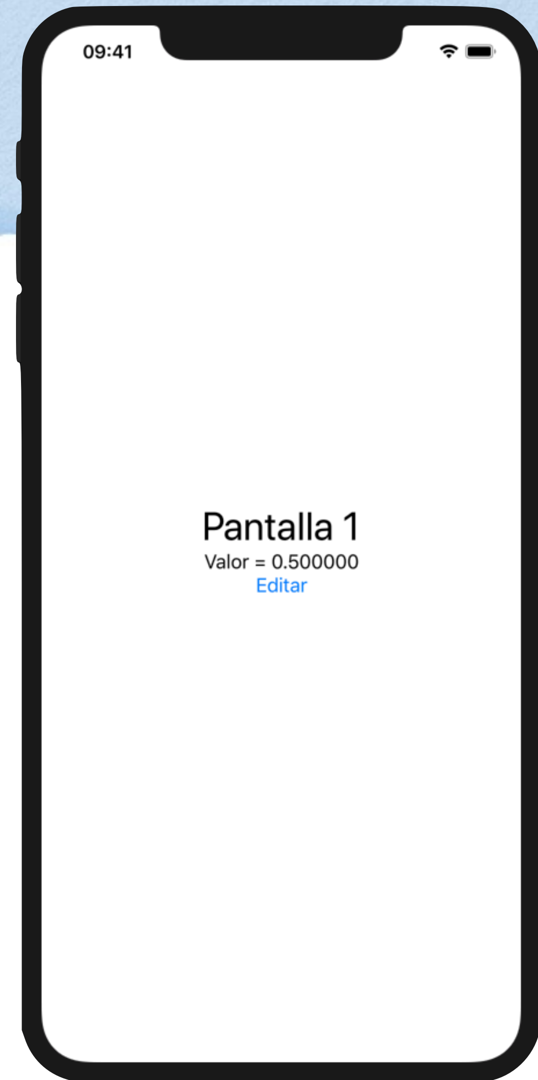
© Santiago Pavón - UPM-DIT

# @Binding

- Enlazar con una propiedad de estado.

© Santiago Pavón - UPM-DIT

# Presentación Modal




```
struct ContentView: View {
    @State var x = 0.5
    @State var showModal = false
    var body: some View {
        VStack {
            Text("Pantalla 1")
                .font(.largeTitle)
            Text("Valor = \(x)")
            Button(action: {
                self.showModal = true
            }) {  Text("Editar") }
        }
        .sheet(isPresented: $showModal) {
            Editor(v: self.$x, showModal: self.$showModal)
        }
    }
}
```

© Santiago Pavón - UPM-DIT

```
struct Editor: View {
    @Binding var v: Double
    @Binding var showModal: Bool

    var body: some View {
        VStack {
            Text("Editor")
                .font(.largeTitle)
            Text("Nuevo Valor = \(v)")
            Slider(value: $v)
            Button(action: {
                self.showModal = false
            }) {  Text("Volver") }
        }
        .padding()
    }
}
```

© Santiago Pavón - UPM-DIT

# Alert

```
struct ContentView: View {

    @State var showAlerta = false

    var body: some View {
        VStack {
            Text("Pantalla")
                .font(.largeTitle)
            Spacer()
            Button(action: {
                self.showAlerta = true
            }) {  Text("Alerta") }
        }
        .alert(isPresented: $showAlerta) {
            Alert(title: Text("Soy una Alerta"),
                  message: Text("Esto es una demo"),
                  dismissButton: .default(Text("cerrar")))
        }
    }
}
```

© Santiago Pavón - UPM-DIT

# @EnvironmentObject

- **@EnvironmentObject** es un property wrapper.
- ¿Cómo funciona?
  1. Si una view tiene una propiedad marcada con el property wrapper **@EnvironmentObject**,
     - por ejemplo:

```
@EnvironmentObject var dato: Dato
```

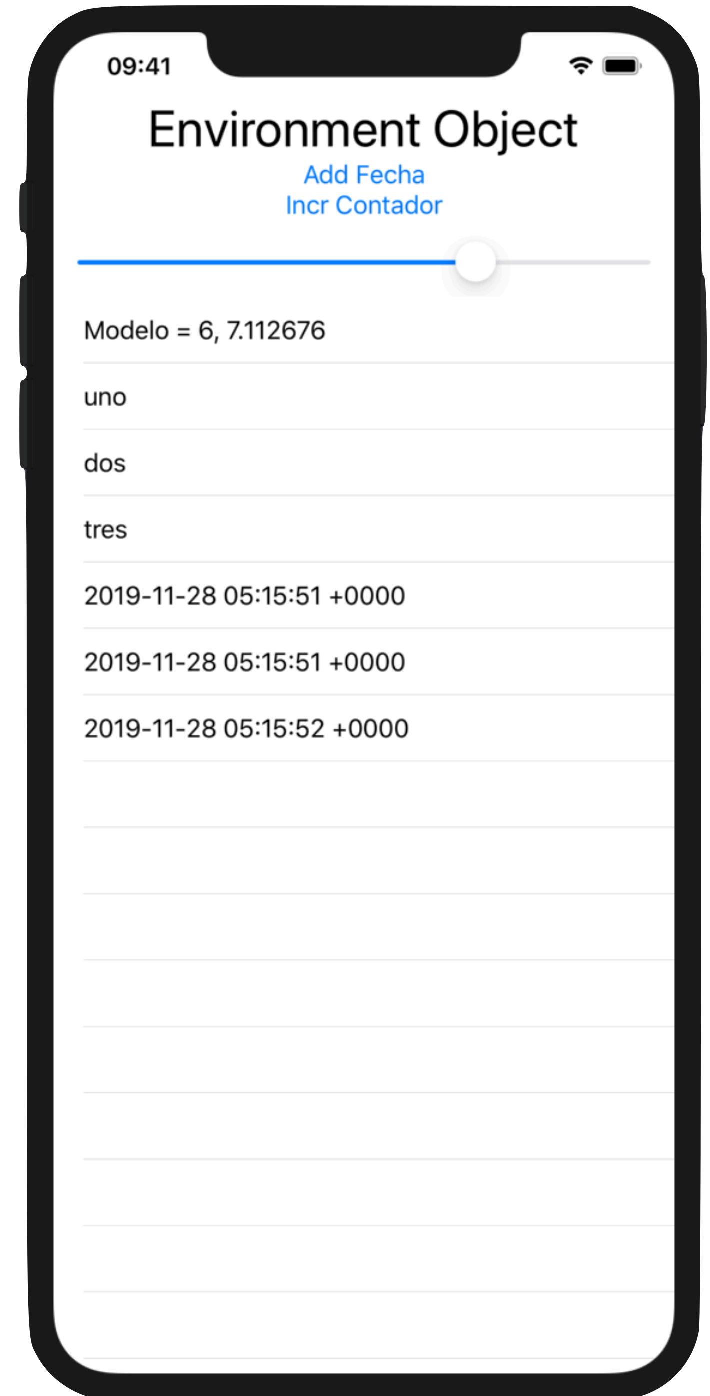
     - Entonces SwiftUI busca en su environment un objeto del tipo **Dato** e inicializa la propiedad **dato** con el objeto encontrado.
       - Si no lo encuentra, la app se muere.
     - Ese objeto (**dato**) es el modelo de nuestras vista.

© Santiago Pavón - UPM-DIT

2. El tipo (en el ejemplo: **Dato**) debe ser una clase conforme con el protocolo **ObservableObject**.
   - SwiftUI se subscribe a los ObservableObject que hay en el environment, y cuando cambian actualiza las views afectadas.
   - En el ejemplo: SwiftUI vigilaría el objeto asignado a **dato** por si hay cambios en él, y repintará la view cada vez que cambie.
3. SwiftUI solo vigila las propiedades de la clase marcada con el property wrapper **@Published**.
   - las views solo se repintan cuando cambia una propiedad marcada con **@Published** del objeto **ObservableObject**.
4. Para meter un objeto **ObservableObject**. en el environment, debe aplicar el modificador **.environmentObject(**dato**)** a una view.
   - El objeto **`ObservableObject`** estará disponible en el environment para esa view, o cualquiera de sus views hijas.
     - ★Ejemplo: Si queremos que el objeto sea el modelo de varias views, es decir, que lo usen varias views, lo más fácil sería crear ese objeto en la clase SceneDelegate donde se crea contentView, y usar el modificador **.environmentObject** con el contentView creado.
5. Los controles de las views que quieran crear un binding con alguna propiedad del objeto **ObservableObject**, deben usar el prefijo **`$`** con el objeto (**dato**).

© Santiago Pavón - UPM-DIT

```swift
import Foundation

class Modelo: ObservableObject {

    @Published var nombres = [
                "uno", "dos", "tres"]

    @Published var contador = 1.0

    func addNombre(_ nombre: String) {
        nombres.append(nombre)
    }
}
```




© Santiago Pavón - UPM-DIT

```
struct ContentView: View {

    @EnvironmentObject var model: Modelo

    var body: some View {
        VStack {
            Text("Environment Object")
                .font(.largeTitle)
            Button(action: {
                self.model.addNombre(Date().description)
            }) {  Text("Add Fecha") }
            Button(action: {
                self.model.contador += 1
            }) {  Text("Incr Contador") }
            Slider(value: self.$model.contador, in: 0.0...10.0)
                .padding(.horizontal)
            List {
                Text("Modelo = \(model.nombres.count), \(model.contador)")
                ForEach(model.nombres, id: \.self) {n in
                    Text(n)
                }
            }
        }
    }
}
struct ContentView_Previews: PreviewProvider {
    static var m = Modelo()
    static var previews: some View {
        ContentView()
        .environmentObject(m)
    }
}
```

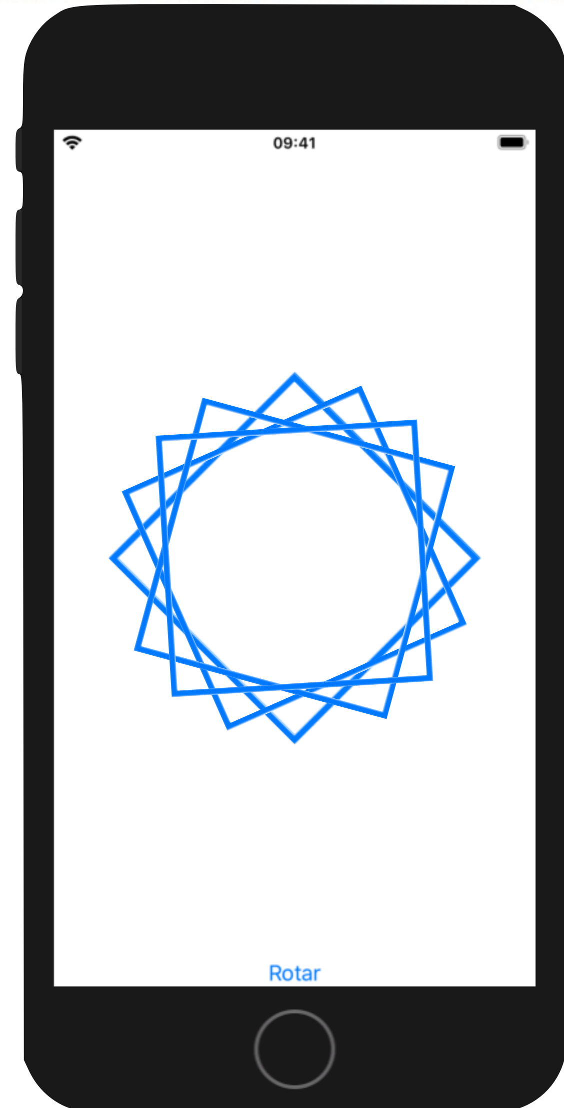
© Santiago Pavón - UPM-DIT

```
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?


    func scene(_ scene: UIScene,
               willConnectTo session: UISceneSession,
               options connectionOptions: UIScene.ConnectionOptions) {

       let contentView = ContentView().environmentObject(Modelo())

       if let windowScene = scene as? UIWindowScene {
            let window = UIWindow(windowScene: windowScene)
            window.rootViewController = UIHostingController(
                                                rootView: contentView)
            self.window = window
            window.makeKeyAndVisible()
        }
    }


. . .
```

© Santiago Pavón - UPM-DIT

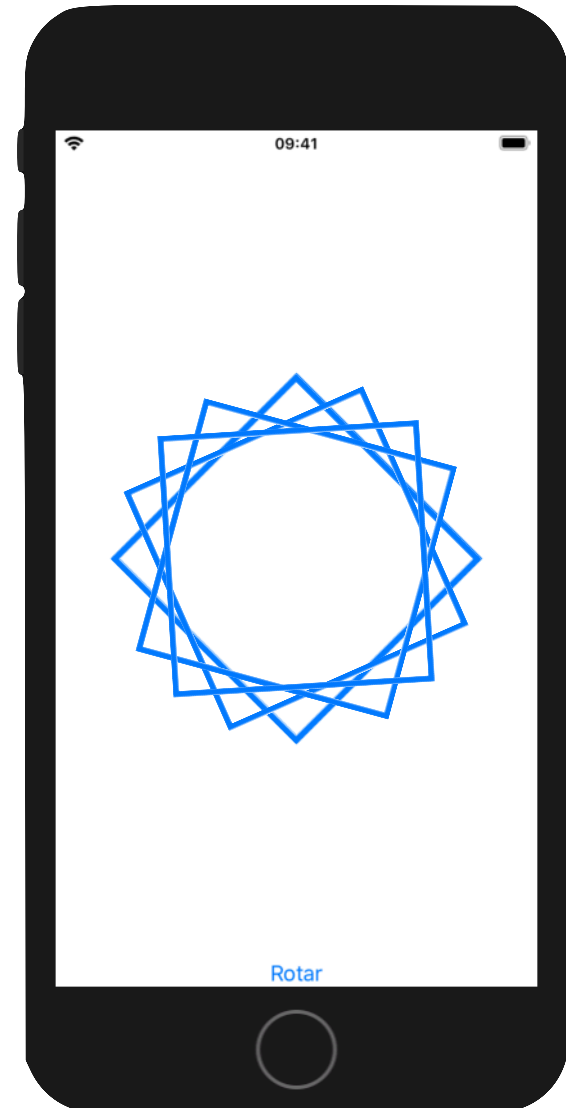
# Animaciones

```
struct ContentView: View {

    @State var r = 45.0

    var body: some View {
        VStack {
            Spacer()

            GeometryReader { geometry in
                Path(CGRect(x: 0, y: 0,
                            width: geometry.size.width,
                            height: geometry.size.height))
                    .stroke(Color.blue, lineWidth: 5)
            }
            .frame(width:200, height: 200)
            .rotationEffect(Angle(degrees: r))
            .animation(.easeInOut)

            Spacer()

            Button(action: {
                self.r = 180 - self.r
            }) { Text("Rotar") }
        }
    }
}
```



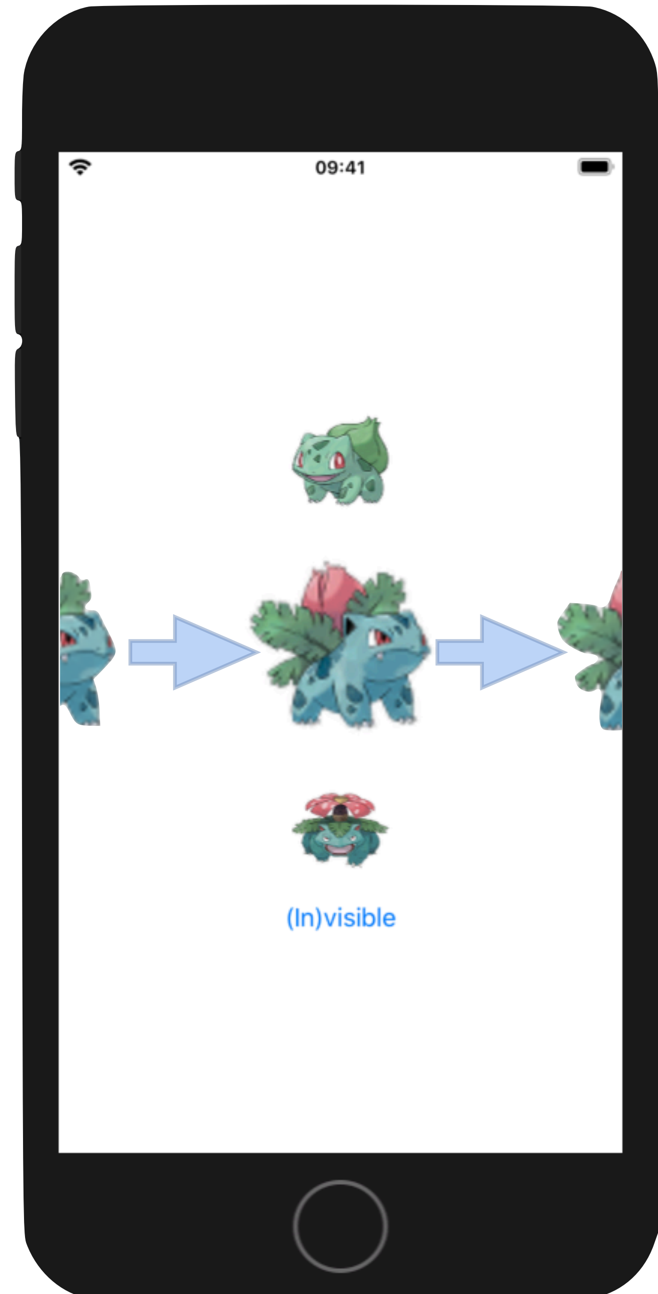

© Santiago Pavón - UPM-DIT

```
struct ContentView: View {

    @State var r = 45.0

    var body: some View {
        VStack {
            Spacer()

            GeometryReader { geometry in
                Path(CGRect(x: 0, y: 0,
                            width: geometry.size.width,
                            height: geometry.size.height))
                    .stroke(Color.blue, lineWidth: 5)
            }
           .frame(width:200, height: 200)
           .rotationEffect(Angle(degrees: r))

            Spacer()

            Button(action: {
              withAnimation {
                    self.r = 180 - self.r
              }
           }) { Text("Rotar") }
        }
    }
}
```




© Santiago Pavón - UPM-DIT

```
struct ContentView: View {

    @State var visible = true

    var body: some View {
        VStack {
            Image("001")
            if self.visible {
                Image("002")
                    .resizable()
                    .scaledToFit()
                    .frame(width: 400, height: 150)
                    .transition(.slide)
            }
            Image("003")

            Button(action: {
                withAnimation {
                    self.visible.toggle()
                }
            }) { Text("(In)visible") }
        }
    }
}
```




© Santiago Pavón - UPM-DIT

# Environment

- SwiftUI almacena en Environment varias configuraciones predefinidas sobre el sistema y las apps.
  - No se guarda cualquier cosa, solo ciertas configuraciones predefinidos.
    - Por ejemplo: el esquema de color, el font por defecto de las apps, ...
- Para leer estas configuraciones se usa el property wrapper **@Environment.**

```
@Environment(\.managedObjectContext) var managedObjectContext

@Environment(\.colorScheme) var colorScheme
```

- Para asignar valores a Environment se usa el modificador **.environment**:

```
.environment(\.font, .body)

.environment(\.multilineTextAlignment, .leading)
```

  - El cambio de configuración afecta a la view donde se aplica, y a todas sus subviews.

© Santiago Pavón - UPM-DIT

- Las keypaths de las configuraciones predefinidos son:

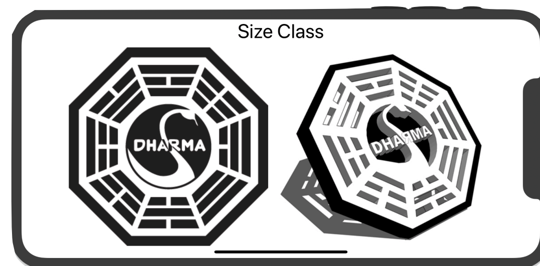
```
.accessibilityDifferentiateWithoutColor .accessibilityEnabled
.accessibilityInvertColors .accessibilityReduceMotion
.accessibilityReduceTransparency .allowsTightening .calendar
.colorScheme .colorSchemeContrast .controlActiveState
.controlSize .defaultMinListHeaderHeight
.defaultMinListRowHeight .defaultWheelPickerItemHeight
.description .disableAutocorrection .displayScale .editMode
.font .horizontalSizeClass .imageScale .isEnabled
.layoutDirection .legibilityWeight .lineLimit .lineSpacing
.locale .managedObjectContext .minimumScaleFactor
.multilineTextAlignment .pixelLength .presentationMode
.sizeCategory .timeZone .truncationMode .undoManager
.verticalSizeClass
```

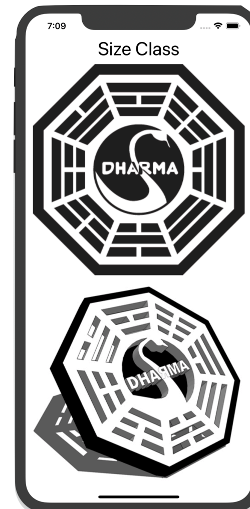
  - https://developer.apple.com/documentation/swiftui/environmentvalues

© Santiago Pavón - UPM-DIT

# Ejemplo: Enviroment y Size Classes

```swift
struct ContentView: View {

    @Environment(\.horizontalSizeClass) var horizontalSizeClass

    var body: some View {
        VStack {
            Text("Size Class")
                .font(.largeTitle)

            if horizontalSizeClass == .compact {
                VStack {
                    Hijo1View()
                    Hijo2View()
                }
            } else {
                HStack {
                    Hijo1View()
                    Hijo2View()
                }
            }
        }
    }
}
```




```swift
struct Hijo1View: View {
    var body: some View {
        Image("dharma1")
            .resizable()
            .scaledToFit()
    }
}

struct Hijo2View: View {
    var body: some View {
        Image("dharma2")
        .resizable()
        .scaledToFit()
    }
}
```

© Santiago Pavón - UPM-DIT

```swift
struct Hijo1View: View {
    var body: some View {
        Image("dharma1")
            .resizable()
            .scaledToFit()
    }
}

struct Hijo2View: View {
    var body: some View {
        Image("dharma2")
        .resizable()
        .scaledToFit()

    }
}
```

© Santiago Pavón - UPM-DIT