

**Sesión 02:**  
**Xcode, Swift, SwiftUI e iOS: Combinación de vistas**  
Grupos de Trabajo  
Líneas de Productos Software

Pablo Gómez Rivas

Departamento de Informática  
Universidad de Almería

Almería, martes 1 de octubre de 2024

## Índice de Contenidos

	Página
Introducción al Lenguaje Swift (II).....	3
Ejercicio 1 .....	3
Ejercicio 2 .....	4
¿Qué ocurre si utilizamos valores por defecto? .....	4
¿Qué sería necesario hacer si nuestro inicializador fuese algo más complejo? .....	4
Ejercicio 3 .....	5
Implementación de Vista Mapa .....	7
Paso 1 .....	7
Paso 2 .....	8
Implementación de Vista Imagen.....	9
Paso 1 .....	9
Paso 2 .....	10
Implementación de Vista Datos (VistaDatos.swift).....	11
Implementación de Vista Detalle (VistaDetalle.swift) .....	12
Paso 1 .....	12
Paso 2 .....	12
Ejercicio 4 .....	12

# Introducción al Lenguaje Swift (II)

## Ejercicio 1



```

1 // Implementación de la función
2 func funcionEjercicio01(valor: Int, accion1: (Int) -> String, accion2: (Int) -> String) {
3     for i in 1...valor {
4         print("Iteración \(i)", "Acción 1", accion1(i), separator: " -> ")
5         print("Iteración \(i)", "Acción 2", accion2(i), separator: " -> ")
6     }
7 }
8
9 // Llamada a la función usando la sintaxis clásica
10 funcionEjercicio01(valor: 3, accion1: { numero in
11     return "Valor \(numero * 2)"
12 }, accion2: { numero in
13     return "Valor \(numero * 3)"
14 })
15
16 // Llamada a la función usando trailing closures
17 funcionEjercicio01(valor: 3) { numero in
18     return "Valor \(numero * 2)"
19 } accion2: { numero in
20     return "Valor \(numero * 3)"
21 }
22
23 // Uso de identificadores predeterminados ($0, $1)
24 funcionEjercicio01(valor: 3) {
25     "Valor \($0 * 2)"
26 } accion2: {
27     "Valor \($0 * 3)"
28 }

```

Iteración 1 -> Acción 1 -> Valor 2  
 Iteración 1 -> Acción 2 -> Valor 3  
 Iteración 2 -> Acción 1 -> Valor 4  
 Iteración 2 -> Acción 2 -> Valor 6  
 Iteración 3 -> Acción 1 -> Valor 6  
 Iteración 3 -> Acción 2 -> Valor 9  
 Iteración 1 -> Acción 1 -> Valor 2  
 Iteración 1 -> Acción 2 -> Valor 3  
 Iteración 2 -> Acción 1 -> Valor 4  
 Iteración 2 -> Acción 2 -> Valor 6  
 Iteración 3 -> Acción 1 -> Valor 6  
 Iteración 3 -> Acción 2 -> Valor 9  
 Iteración 1 -> Acción 1 -> Valor 2  
 Iteración 1 -> Acción 2 -> Valor 3  
 Iteración 2 -> Acción 1 -> Valor 4  
 Iteración 2 -> Acción 2 -> Valor 6  
 Iteración 3 -> Acción 1 -> Valor 6  
 Iteración 3 -> Acción 2 -> Valor 9

## Ejercicio 2

Ejercicio2

EjerciciosSesion02 > Ejercicio2 > No Selection

```
1 import UIKit
2
3 // Definición de la función getMessage
4 func getMessage(lugar: String, accion: (Int, String) -> String) -> String {
5     // Llama a la clausura con un número aleatorio entre 1 y 3, y el lugar proporcionado
6     return accion(Int.random(in: 1...3), lugar)
7 }
8
9 // Llamada a la función getMessage con la clausura definida
10 var mensaje: String = getMessage(lugar: "Londres") { (tipoViaje, lugar) -> String in
11     switch tipoViaje {
12     case 1:
13         return "me voy de marcha a \(lugar)"
14     case 2:
15         return "me voy a trabajar a \(lugar)"
16     case 3:
17         return "me voy a estudiar a \(lugar)"
18     default:
19         return "me voy a \(lugar)"
20     }
21 }
22
23 // Imprimir el mensaje
24 print(mensaje)
25
26 // Llamada a la función getMessage simplificada usando valores predeterminados
27 var mensajeSimplificado: String = getMessage(lugar: "Londres") { "me voy \(($0 == 1) ? "de marcha" : ($0 == 2) ? "a trabajar" :
28     ($0 == 3) ? "a estudiar" : "")" } a \(\$1)" }
29
30 // Imprimir el mensaje simplificado
31 print(mensajeSimplificado)
```

me voy a estudiar a Londres  
me voy de marcha a Londres

¿Qué ocurre si utilizamos valores por defecto?

En Swift, si se usan valores por defecto en una estructura, se puede inicializar objetos sin especificar todos los parámetros. Ejemplo:

```
struct Point2D {  
    var componentX: Int = 0  
    var componentY: Int = 0  
    func toString() -> String {  
        return "(\(componentX), \(componentY))"  
    }  
}
```

Esto te permite crear instancias como:

```
let punto1 = Point2D() // (0, 0)  
let punto2 = Point2D(componentX: 10) // (10, 0)
```

¿Qué sería necesario hacer si nuestro inicializador fuese algo más complejo?

Si necesitas lógica más compleja en el inicializador, se puede personalizarlo con validaciones:

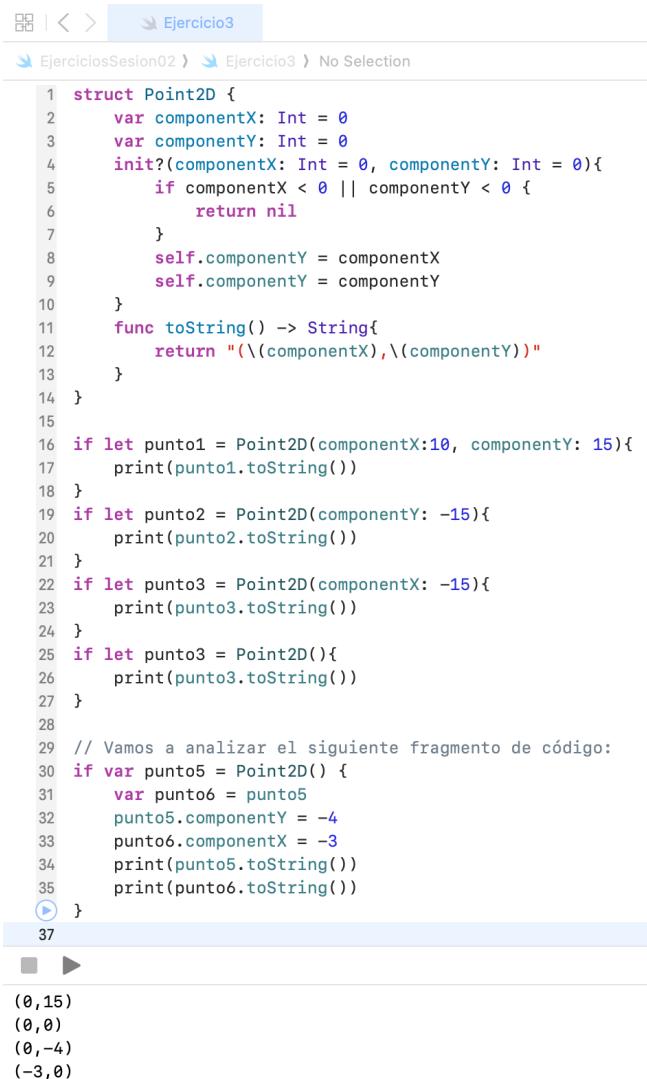
```
init?(componentX: Int, componentY: Int) {  
    guard componentX >= 0, componentY >= 0 else { return nil }  
    self.componentX = componentX
```

```

    self.componentY = componentY
}

```

### Ejercicio 3



```

1 struct Point2D {
2     var componentX: Int = 0
3     var componentY: Int = 0
4     init?(componentX: Int = 0, componentY: Int = 0){
5         if componentX < 0 || componentY < 0 {
6             return nil
7         }
8         self.componentX = componentX
9         self.componentY = componentY
10    }
11    func toString() -> String{
12        return "(\(componentX), \(componentY))"
13    }
14 }
15
16 if let punto1 = Point2D(componentX:10, componentY: 15){
17    print(punto1.toString())
18 }
19 if let punto2 = Point2D(componentY: -15){
20    print(punto2.toString())
21 }
22 if let punto3 = Point2D(componentX: -15){
23    print(punto3.toString())
24 }
25 if let punto3 = Point2D(){
26    print(punto3.toString())
27 }
28
29 // Vamos a analizar el siguiente fragmento de código:
30 if var punto5 = Point2D() {
31     var punto6 = punto5
32     punto5.componentY = -4
33     punto6.componentX = -3
34     print(punto5.toString())
35     print(punto6.toString())
36 }
37
38
(0,15)
(0,0)
(0,-4)
(-3,0)

```

En Swift, las estructuras se comportan como tipos por valor. Esto significa que cuando asignas una estructura a otra variable (como punto6 = punto5), se crea una copia independiente. Por lo tanto, las modificaciones a una copia no afectan a la otra. Por defecto, las propiedades de una estructura como Point2D son públicas para su acceso y modificación si no se indica lo contrario con modificadores de acceso como private o fileprivate.

#### Inicialización de punto5 y punto6:

- Point2D() crea una instancia con los valores predeterminados de componentX = 0 y componentY = 0.
- Se verifica si la inicialización es exitosa (ya que el inicializador es opcional debido a que puede fallar si los valores son negativos).
- Se asigna una copia de punto5 a punto6. Dado que Point2D es una estructura, estamos trabajando con copias de los valores y no con referencias al mismo objeto.

#### Modificación de punto5 y punto6:

- punto5.componentY = -4: Cambia el valor de componentY de punto5 a -4.
- punto6.componentX = -3: Cambia el valor de componentX de punto6 a -3.

- Como punto5 y punto6 son copias (por ser estructuras), estas modificaciones no afectan a la otra variable. Cada una mantiene su propio estado independiente.

### Salida por consola:

- `print(punto5.toString())` imprime "(0,-4)" porque punto5 tiene componentX = 0 y componentY = -4.
- `print(punto6.toString())` imprime "(-3,0)" porque punto6 tiene componentX = -3 y componentY = 0.

### Gestión dinámica vs. estática de memoria (Heap vs. Stack)

- Gestión estática de memoria (Stack):

En el Stack, la memoria se asigna de forma automática y temporal, y los datos se liberan automáticamente cuando ya no se necesitan.

Esto sucede principalmente con tipos por valor, como las estructuras en Swift. Cada vez que se crea una nueva instancia de una estructura, se coloca en el Stack. Como el Stack sigue un orden LIFO (Last In, First Out), la memoria se libera automáticamente cuando las variables salen del ámbito de la función.

Es muy eficiente, pero el Stack tiene un tamaño limitado.

- Gestión dinámica de memoria (Heap):

El Heap es una región de la memoria donde se asigna espacio de manera dinámica, principalmente para tipos por referencia, como las clases en Swift.

A diferencia del Stack, la memoria asignada en el Heap no se libera automáticamente. Los objetos almacenados en el Heap deben gestionarse mediante un sistema de control de memoria, como el conteo de referencias (ARC en Swift), que libera la memoria cuando ya no hay referencias activas a un objeto.

Aunque el Heap es más flexible y puede manejar grandes cantidades de datos, la gestión de la memoria en el Heap es más lenta que en el Stack.

# Implementación de Vista Mapa

## Paso 1

The screenshot shows the Xcode interface with two code editors and a preview area.

**Left Editor:** Displays the main file `GestionAmigos/main`. It contains the following SwiftUI code:

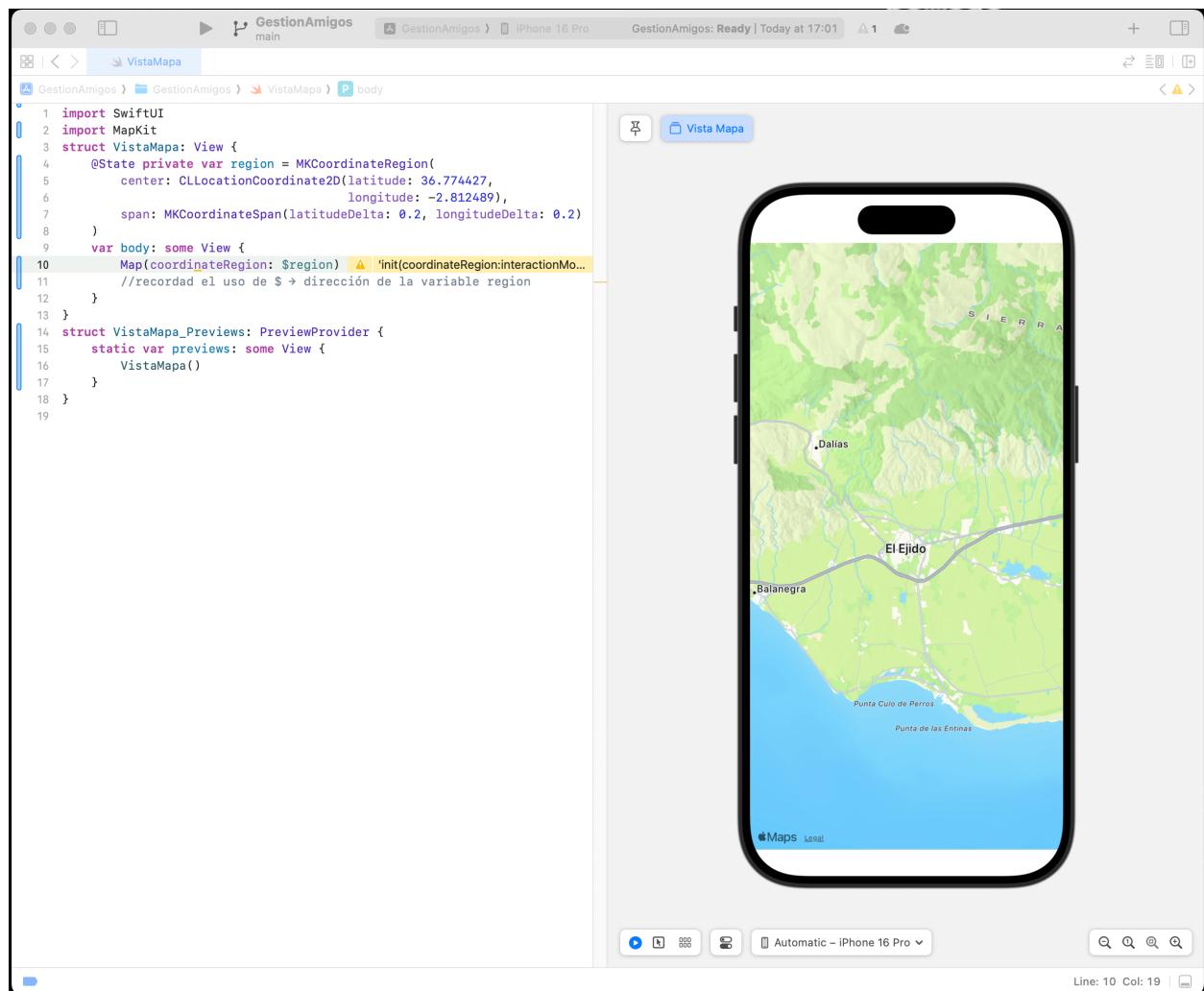
```
1 import SwiftUI
2
3 @main
4 struct GestionAmigosApp: App {
5     var body: some Scene {
6         WindowGroup {
7             VistaMapa()
8         }
9     }
10 }
11
12 }
```

**Right Editor:** Displays a file named `VistaMapa`. It contains the following SwiftUI code:

```
1 import SwiftUI
2 import MapKit
3
4 struct VistaMapa: View {
5     var body: some View {
6         Text("Hello, World!")
7     }
8 }
9
10 #Preview {
11     VistaMapa()
12 }
```

**Preview Area:** Shows a simulated iPhone 16 Pro screen with a black border. The screen displays the text "Hello, World!" in a simple font.

## Paso 2

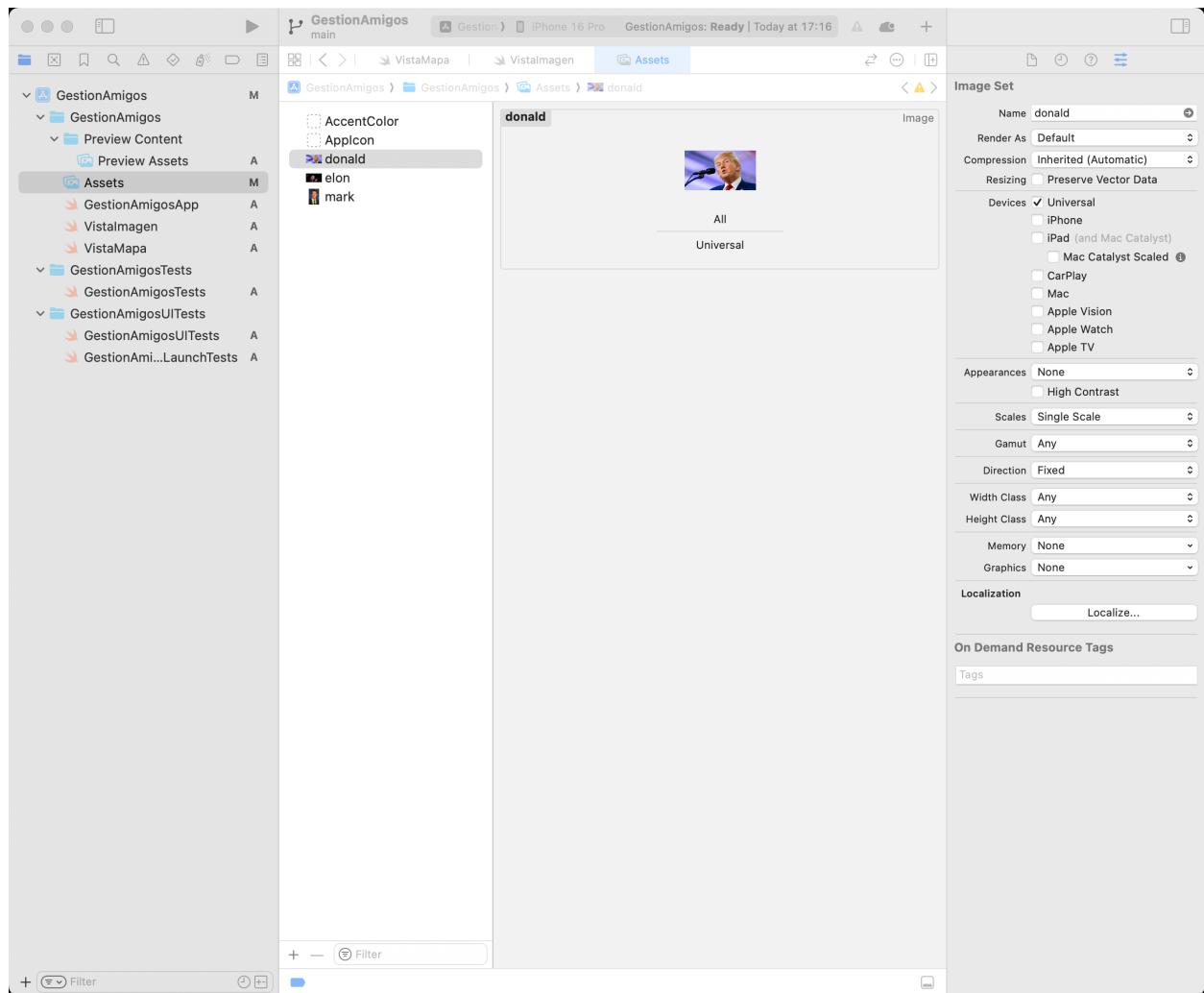


The screenshot shows the Xcode interface with the following details:

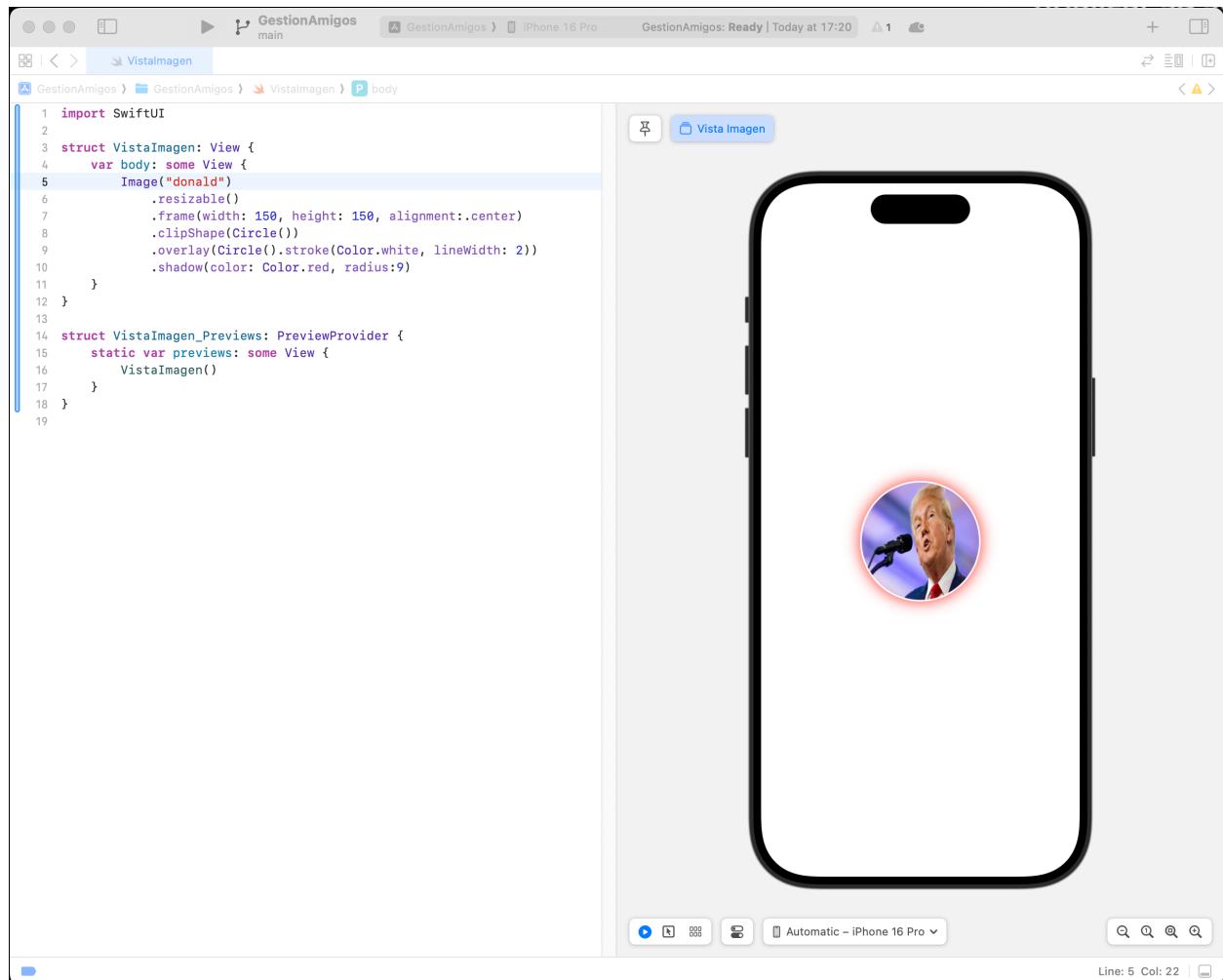
- Project Structure:** GestionAmigos > GestionAmigos > VistaMapa > body
- Code Editor:** Shows Swift code for a MapKit view. The code defines a struct `VistaMapa` with a private `region` variable set to an `MKCoordinateRegion` with center coordinates `36.774427, -2.812489` and a `span` of `0.2`. It also includes a `body` property which is a `Map` view initialized with the region. A note in the code says `//recordad el uso de $ & dirección de la variable region`.
- Preview Area:** Shows a preview of the `VistaMapa` view on an iPhone 16 Pro. The map displays a coastal area with green terrain, blue water, and several labeled locations: `Dalias`, `El Ejido`, `Balanegra`, `Punta Culo de Perros`, and `Punta de las Entinas`. A road network is also visible.
- Bottom Status Bar:** Shows "Automatic – iPhone 16 Pro".
- Bottom Right:** Includes status icons for battery, signal, and volume, along with a search bar and a line number indicator "Line: 10 Col: 19".

# Implementación de Vista Imagen

## Paso 1



## Paso 2



The screenshot shows the Xcode interface with the following details:

- Project:** GestiónAmigos
- File:** main
- Preview View:** VistaImagen
- Preview Device:** iPhone 16 Pro
- Preview Title:** Vista Imagen
- Code Preview:** Shows the SwiftUI code for a view named `VistaImagen` which displays a circular image of Donald Trump.

```
1 import SwiftUI
2
3 struct VistaImagen: View {
4     var body: some View {
5         Image("donald")
6             .resizable()
7             .frame(width: 150, height: 150, alignment: .center)
8             .clipShape(Circle())
9             .overlay(Circle().stroke(Color.white, lineWidth: 2))
10            .shadow(color: Color.red, radius:9)
11    }
12 }
13
14 struct VistaImagen_Previews: PreviewProvider {
15     static var previews: some View {
16         VistaImagen()
17     }
18 }
```

- Preview Content:** An iPhone 16 Pro simulator displaying a circular portrait of Donald Trump with a red shadow effect.
- Bottom Bar:** Includes icons for play, stop, and zoom, and a dropdown menu for device selection.
- Status Bar:** Shows "Automatic - iPhone 16 Pro".
- Bottom Right:** Includes search and refresh icons.
- Bottom Left:** Shows "Line: 5 Col: 22" and a small icon.

# Implementación de Vista Datos (VistaDatos.swift)

The screenshot shows the Xcode interface with the code editor open to the file `VistaDatos.swift`. The code defines a `struct` `VistaDatos` that displays information about Donald Trump, including his name, phone number, email, and a bio. A preview of the view is shown on the right, displaying a green background with white text and icons.

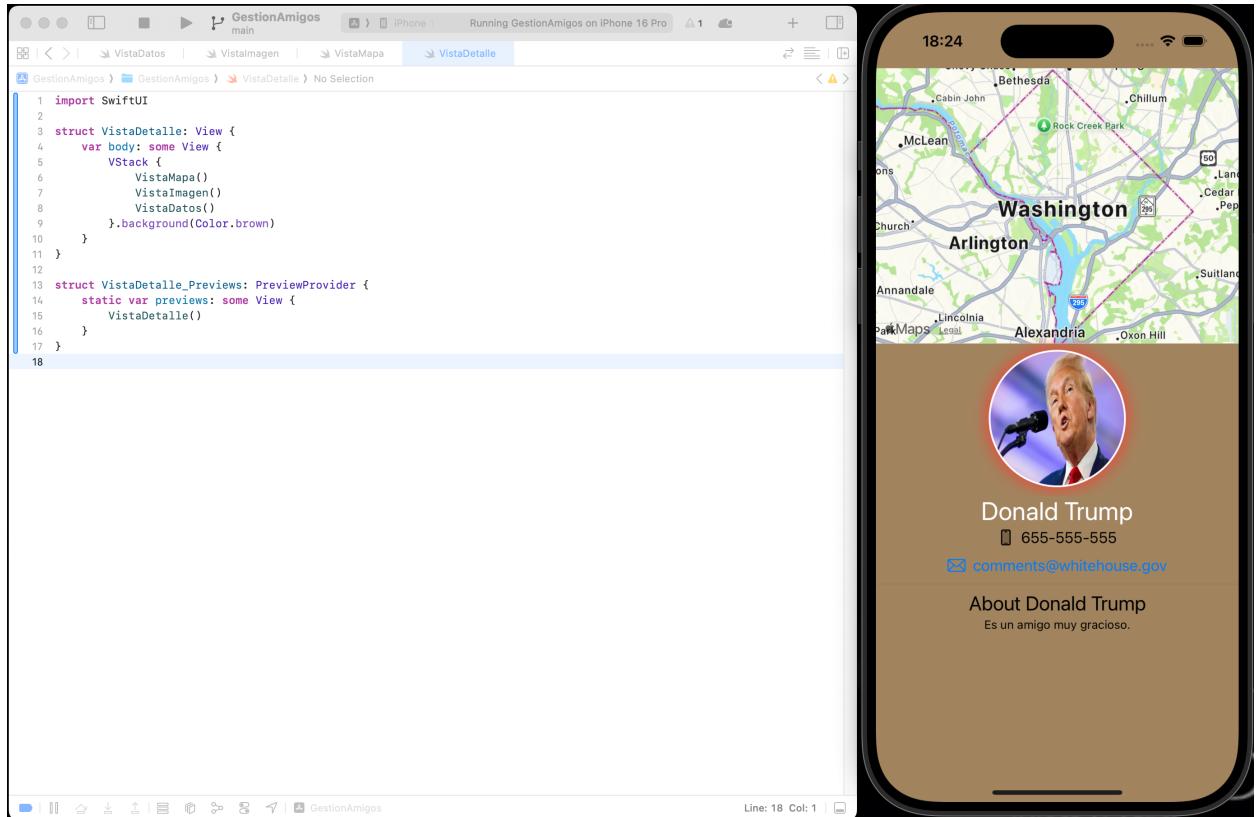
```

1 import SwiftUI
2
3 struct VistaDatos: View {
4     var body: some View {
5         VStack {
6             Text("Donald Trump")
7                 .font(.title)
8                 .foregroundColor(.white)
9             Label("655-555-555", systemImage:"iphone")
10            .font(.body)
11            Link(destination: URL(string:
12                "mailto:donaldtrump@gmailing.com")!, label: {
13                Image(systemName: "livephoto")
14                    .frame(width: 20, height: 20,
15                        alignment: .center)
16                Text("donaldtrump@gmailing.com")
17            })
18            Divider()
19            Text("About Donald Trump")
20                .font(.title2)
21            Text("Es un amigo muy gracioso.")
22                .font(.footnote)
23            Spacer()
24        }.background(Color.green)
25    }
26
27 struct VistaDatos_Previews: PreviewProvider {
28     static var previews: some View {
29         VistaDatos()
30     }
31 }

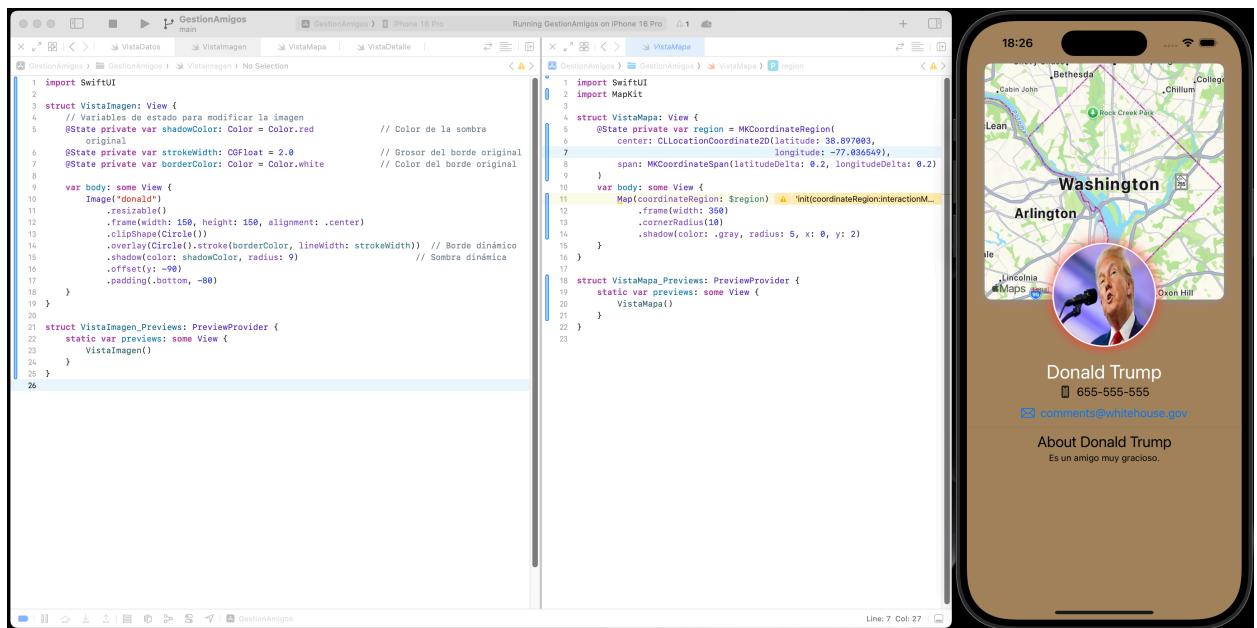
```

# Implementación de Vista Detalle (VistaDetalle.swift)

## Paso 1

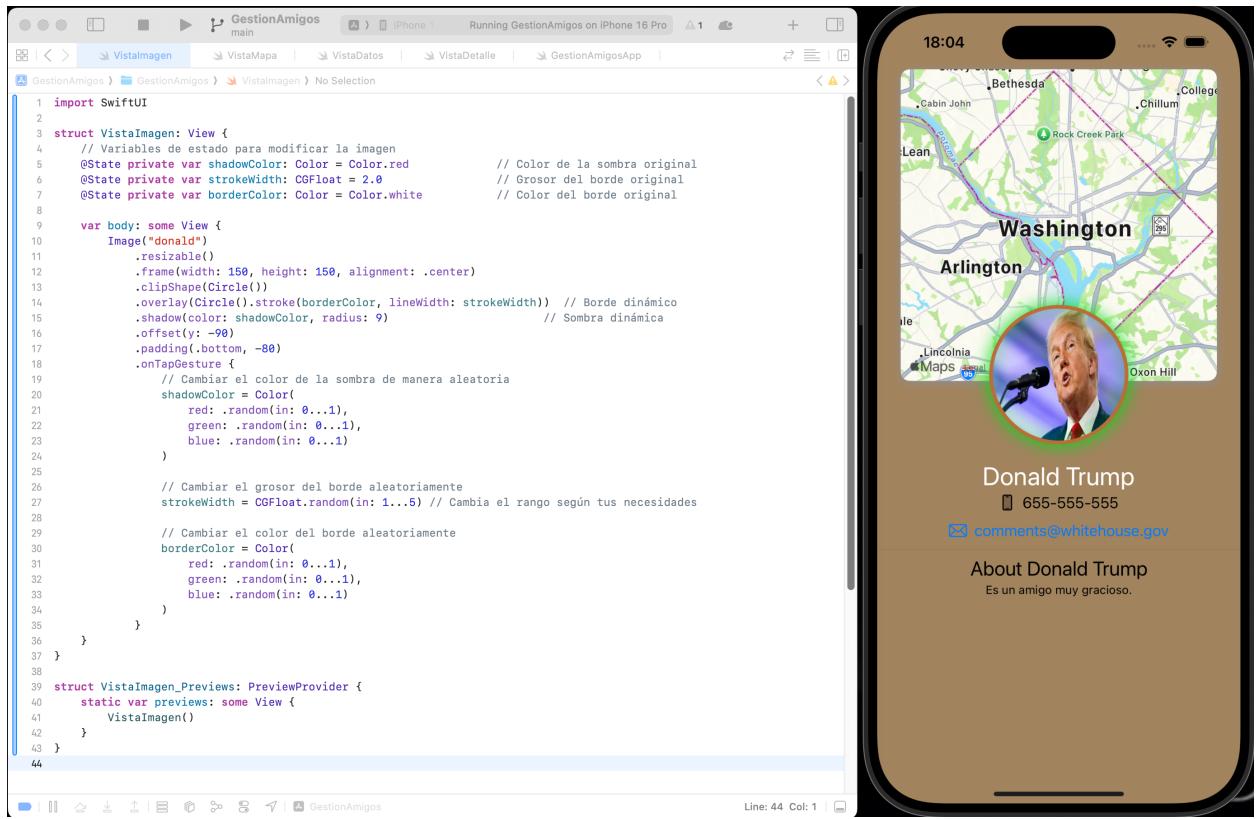


## Paso 2



## Ejercicio 4

Al hacer tap en la foto del amigo, esta cambia su color de la sombra, color del borde, y grosor del borde de manera aleatoria.



The image shows a developer's workspace. On the left, the Xcode interface displays the Swift code for the `VistaImagen` view. The code defines a `VistaImagen` struct that takes a `View` parameter named `body`. It applies a `resizable()` modifier to the body. The body itself is an `Image("donald")` with a `frame(width: 150, height: 150, alignment: .center)`. It uses `clipShape(Circle())` and `overlay(Circle().stroke(borderColor, lineWidth: strokeWidth))` to create a circular border. The `shadow` modifier is used with a `radius: 9` and a `color: shadowColor`. A `padding(.bottom, -80)` is applied. An `onTapGesture` block changes the `shadowColor` to a random color between red, green, and blue. The `strokeWidth` is set to a random value between 1 and 5. The `VistaImagen_Previews` section contains a static preview for `VistaImagen()`. On the right, an iPhone 16 Pro mockup shows a map of Washington D.C. with a callout for "Donald Trump". The callout features a circular profile picture of Donald Trump speaking into a microphone, with a green glow around it. Below the picture, the text "Donald Trump" and the phone number "655-555-555" are displayed, along with an email link "comments@whitehouse.gov". The callout also includes the text "About Donald Trump" and "Es un amigo muy gracioso."

```

1 import SwiftUI
2
3 struct VistaImagen: View {
4     // Variables de estado para modificar la imagen
5     @State private var shadowColor: Color = Color.red           // Color de la sombra original
6     @State private var strokeWidth: CGFloat = 2.0              // Grosor del borde original
7     @State private var borderColor: Color = Color.white         // Color del borde original
8
9     var body: some View {
10        Image("donald")
11            .resizable()
12            .frame(width: 150, height: 150, alignment: .center)
13            .clipShape(Circle())
14            .overlay(Circle().stroke(borderColor, lineWidth: strokeWidth)) // Borde dinámico
15            .shadow(color: shadowColor, radius: 9)                         // Sombra dinámica
16            .offset(y: -90)
17            .padding(.bottom, -80)
18            .onTapGesture {
19                // Cambiar el color de la sombra de manera aleatoria
20                shadowColor = Color(
21                    red: .random(in: 0...1),
22                    green: .random(in: 0...1),
23                    blue: .random(in: 0...1)
24                )
25
26                // Cambiar el grosor del borde aleatoriamente
27                strokeWidth = CGFloat.random(in: 1...5) // Cambia el rango según tus necesidades
28
29                // Cambiar el color del borde aleatoriamente
30                borderColor = Color(
31                    red: .random(in: 0...1),
32                    green: .random(in: 0...1),
33                    blue: .random(in: 0...1)
34                )
35            }
36        }
37    }
38
39 struct VistaImagen_Previews: PreviewProvider {
40     static var previews: some View {
41         VistaImagen()
42     }
43 }
44

```