

Vistas, estructuras y propiedades

Personalizar las vistas con propiedades



Crea una aplicación SwiftUI creando vistas personalizadas para hacer un pronóstico del tiempo de varios días. En tu vista personalizada, usarás propiedades para personalizar la pantalla para cada día.

Diseñas vistas personalizadas utilizando estructuras. Las estructuras son una forma de organizar su código para que las piezas relacionadas se empaqueten juntas. Cada estructura tiene un nombre que te permite reutilizarla como una plantilla en cualquier lugar de tu aplicación. Con las estructuras, puedes escribir código que sea más eficiente y tenga menos errores.

Prototipo de una vista de pronóstico

Crea una interfaz sencilla para mostrar dos pronósticos meteorológicos.

Fíjate en el patrón repetido en la interfaz.

Mon	Tue
	
High: 70	High: 60
Low: 50	Low: 40

Paso 1

En ContentView, elimine el código dentro del VStack y reemplácelo con un pronóstico del tiempo simple.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {

            Text("Mon")
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Mon
High: 70
Low: 50

Paso 2

Añade una vista de imagen a tu pronóstico y utilízala para mostrar un símbolo de SF de un sol.

Este tipo de imagen requiere un argumento, cuya etiqueta de argumento es `systemName`.

Nota

Descarga la aplicación [SF Symbols](#) para navegar por miles de imágenes que puedes usar en toda una aplicación.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Mon
☀️
High: 70
Low: 50

Paso 3

Usa el modificador `.foregroundColor` para hacer que el icono del sol sea amarillo.

Utilizas el modificador `.foregroundColor` para aplicar un color a los elementos de primer plano de cualquier vista. Por ejemplo, establecer un color como el estilo de primer plano de una vista de texto cambia el color del texto.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}

#Preview {
    ContentView()
}
```

Mon
☀️
High: 70
Low: 50

Paso 4

Prepárate para añadir un segundo pronóstico haciendo clic con el botón derecho en VStack y eligiendo Insertar en HStack.

Un HStack es una vista de contenedor que organiza las vistas horizontalmente. Un VStack organiza las vistas que contiene verticalmente.

Nota

No hay ningún cambio en la interfaz, porque el HStack solo contiene un elemento.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            VStack {

                Text("Mon")
                Image(systemName: "sun.max.fill")
                    .foregroundColor(Color.yellow)
                Text("High: 70")
                Text("Low: 50")

            }
            .padding()
        }
    }
}

#Preview {
    ContentView()
}
```

Mon
☀️
High: 70
Low: 50



Paso 5

Copie todo el VStack, incluido el modificador .padding, y luego pegue el código debajo del primer pronóstico para crear una segunda vista de pronóstico. Actualice los datos para mostrar el pronóstico del día siguiente.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            VStack {
                Text("Mon")
                Image(systemName: "sun.max.fill")
                    .foregroundColor(Color.yellow)
                Text("High: 70")
                Text("Low: 50")
            }
            .padding()
            VStack {
                Text("Tue")
                Image(systemName: "cloud.rain.fill")
                    .foregroundColor(Color.blue)
                Text("High: 60")
                Text("Low: 40")
            }
            .padding()
        }
    }
}

#Preview {
    ContentView()
}
```

Mon	Tue
	
High: 70	High: 60
Low: 50	Low: 40

Crea una subvista personalizada

En lugar de seguir copiando y pegando su código, aprenda a encapsular el pronóstico de un día en su propia estructura para hacer una tarjeta de pronóstico personalizable.

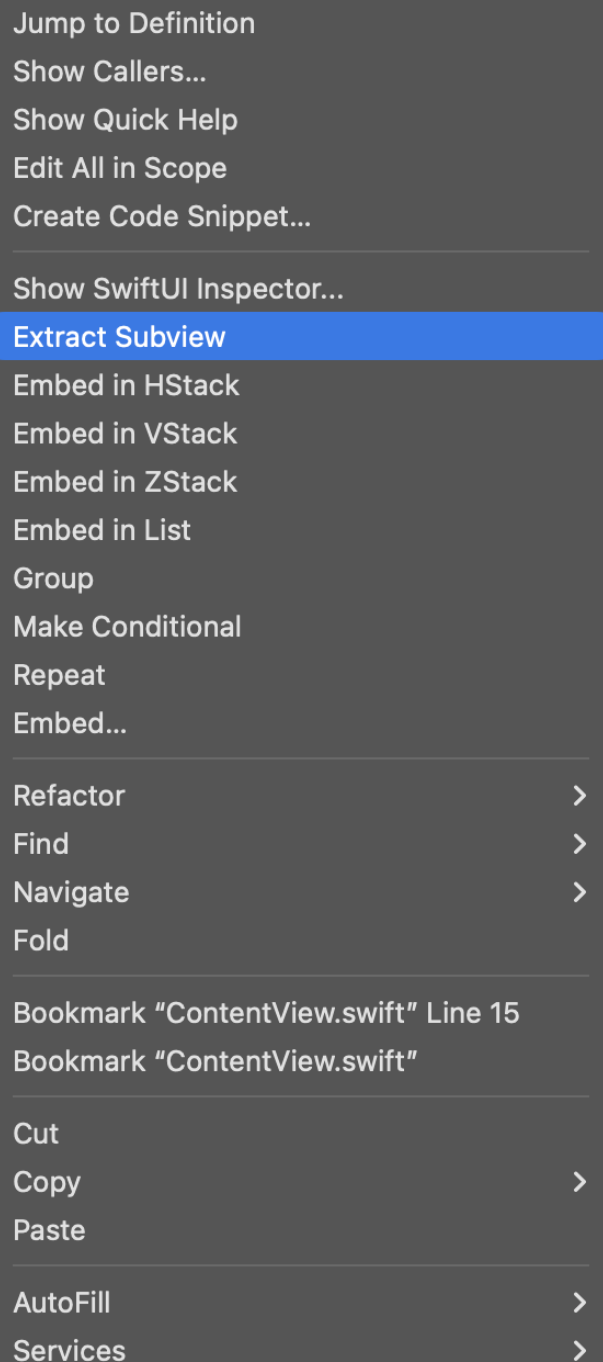
Paso 1

Haga clic con el botón derecho en el VStack para ver el pronóstico del primer día. Xcode presenta un menú de acciones comunes. Elija Extraer Subvista.

Xcode reemplaza el VStack con ExtractedView(). Mire más abajo en su código para encontrar su nueva vista personalizada, struct ExtractedView.

Nota

Una subvista es una vista que se utiliza dentro de otra vista. Por ejemplo, en el código predeterminado de un nuevo proyecto, el texto y la imagen son subvistas de ContentView.



Paso 2

En ambas ubicaciones, cambio el nombre de `ExtractedView` a `DayForecast`. Ahora, en cualquier lugar que escribas `DayForecast()`, estás usando todo el código que extrajiste en la nueva vista.

Ahora que has creado una vista de `DayForecast`, puedes usarla para el pronóstico de cada día.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast()



            VStack {
                Text("Tue")
                Image(systemName: "cloud.rain.fill")
                    .foregroundColor(Color.blue)
                Text("High: 60")
                Text("Low: 40")
            }
                .padding()
        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Mon	Tue
	
High: 70	High: 60
Low: 50	Low: 40

Paso 3

Reemplace el segundo pronóstico por otra vista de DayForecast.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast()

            DayForecast()

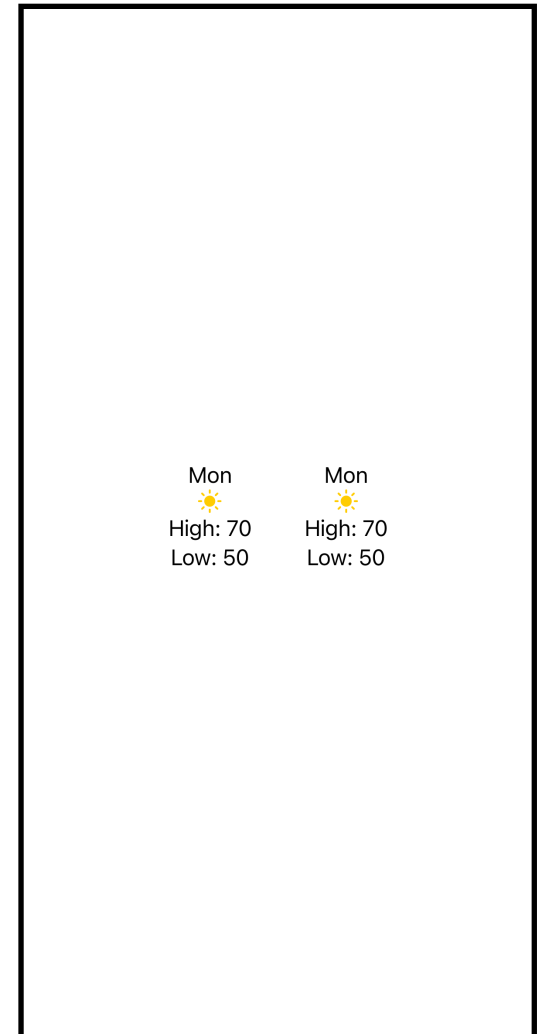
        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```



Paso 4

Examina tu código. Hay tres lugares en los que hace referencia a DayForecast. Dentro de HStack, estás creando dos instancias de DayForecast para mostrarlas en la pantalla. En la parte inferior, estás definiendo la vista y su contenido.

Nota

Usted define o declara - una estructura usando la palabra clave struct.

Utilizas o creas una instancia de la estructura escribiendo su nombre seguido de paréntesis.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast()

            DayForecast()

        }
    }
}



#Preview {
    ContentView()
}

struct DayForecast: View {

    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Mon	Mon
	
High: 70	High: 70
Low: 50	Low: 50

Generalizar el día con una propiedad

Puede usar propiedades para mostrar cualquier dato de pronóstico utilizando su vista personalizada. Empieza por hacer que el día de la semana sea personalizable.

Paso 1

Añade una nueva propiedad a la vista DayForecast para almacenar datos sobre el día de la semana. Añade una línea en blanco entre la propiedad y la propiedad del cuerpo debajo de ella.

Puedes usar líneas en blanco en Swift para hacer que tu código sea más legible; no tienen ningún impacto en la forma en que se ejecuta tu código.

Importante

Xcode muestra un error, que corregirás en un momento.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast()

            DayForecast()

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Paso 2

Antes de corregir el error, eche un vistazo a la propiedad que acaba de crear.

Para declarar una propiedad, utilice la palabra clave `let`. Una propiedad tiene un nombre, seguido de dos puntos y luego el tipo de datos que contiene. El nombre de esta propiedad es `day`, y tiene una cadena.

Nota

En Swift, los tipos de datos se llaman tipos. Puedes leer el código que agregaste a la estructura de `DayForecast` de la siguiente manera: "day es una propiedad cuyo tipo es `String`". O podrías decir: "el día es una propiedad de tipo `String`".

Nombre

Tipo

```
let day: String
```

Paso 3

Mira el error en el que usas DayForecast por primera vez: "Argumento que falta para el parámetro 'día' en la llamada". Haga clic en el icono junto al mensaje de error en Xcode para expandirlo.

Cuando agregas una propiedad a una estructura, debes darle un valor cada vez que creas una instancia de la estructura. Este error significa que no le has dado un valor a la propiedad del día.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast()

            DayForecast()

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Paso 4

Haga clic en el botón Fix en el banner de error para agregar la etiqueta del argumento, día:.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: String)

            DayForecast()

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Paso 5

Reemplace el marcador de posición con un día de la semana entre comillas.

Estás llamando al inicializador de `DayForecast` y pasando un argumento `String` para el parámetro `day`.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon")

            DayForecast()

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Paso 6

Corrige el segundo error de la misma manera, usando un día diferente para el argumento.

La vista previa no cambia, porque todavía estás pasando una cadena literal en la primera vista de texto en la estructura DayForecast.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon")

            DayForecast(day: "Tue")

        }
    }
}



#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    var body: some View {
        VStack {

            Text("Mon")
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Mon	Mon
	
High: 70	High: 70
Low: 50	Low: 50

Paso 7

Utilice la propiedad `day` en la vista `Texto` de su estructura `DayForecast` para que su vista previa refleje los datos que agregó anteriormente.

La propiedad `day` tiene un valor de cadena. Al usar la propiedad como argumento, estás pasando su valor a la vista `Texto`.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon")

            DayForecast(day: "Tue")

        }
    }
}

#Preview {
    ContentView()
}



struct DayForecast: View {
    let day: String

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

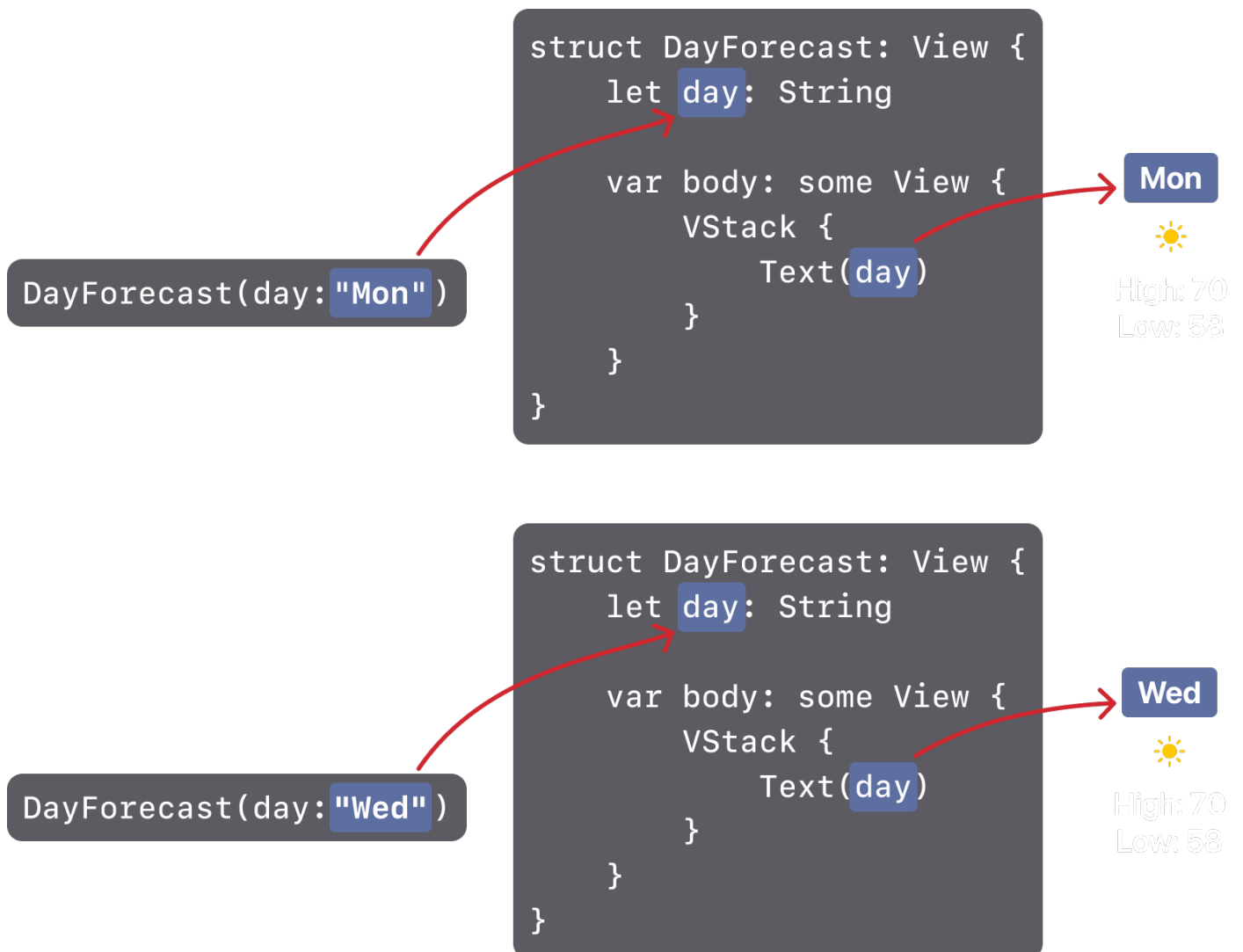
        }
        .padding()
    }
}
```

Mon	Tue
	
High: 70	High: 70
Low: 50	Low: 50

Paso 8

Echa un vistazo a cómo el valor de la cadena viaja desde tu código ContentView a una instancia de DayForecast y desde allí hasta tu interfaz.

Has generalizado DayForecast para mostrar el nombre de cualquier día.



Usa Int para mostrar las temperaturas

Extienda DayForecast para generalizar el rango de temperatura. Agregue propiedades para representar las temperaturas como números, usando un nuevo tipo, Int, luego use la interpolación de cadenas para mostrar las temperaturas en una vista de texto.

Paso 1

Almacene las temperaturas altas y bajas como propiedades Int en DayForecast. Int es un tipo de datos que representa un número entero.

Tienes errores como los que hiciste cuando agregaste la propiedad day, porque a tus inicializadores de DayForecast ahora les faltan argumentos.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon")

            DayForecast(day: "Tue")

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    let high: Int

    let low: Int

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Paso 2

Corrige los errores actualizando las dos instancias de DayForecast con datos de temperatura. Cuando hay dos o más argumentos, los separas con comas.

Como antes, la vista previa no cambiará hasta que utilices las nuevas propiedades en la estructura DayForecast.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", high: 70, low: 50)

            DayForecast(day: "Tue", high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let high: Int
    let low: Int
    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("High: 70")
            Text("Low: 50")

        }
        .padding()
    }
}
```

Paso 3

En DayForecast, actualice los inicializadores de la vista de texto para los dos valores de temperatura.

Aparece otro error, "No hay coincidencias exactas en la llamada al inicializador", porque no hay forma de inicializar una vista de texto con un valor numérico. Necesitas una forma de convertir un número en una cadena para pasar al inicializador de texto. Puedes usar la interpolación de cadenas para hacer precisamente eso.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", high: 70, low: 50)

            DayForecast(day: "Tue", high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let high: Int
    let low: Int
    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)

            Text(high)

            Text(low)

        }
        .padding()
    }
}
```

Paso 4

Cambie los valores `Int` en los inicializadores de texto a literales de cadena, utilizando la interpolación para insertar los valores de sus dos propiedades de temperatura.

Si desea mostrar el valor de un número en una vista de texto, que espera una cadena, tiene que cambiar el número por una cadena, lo que le gusta: `"\(<number>)"`.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", high: 70, low: 50)

            DayForecast(day: "Tue", high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let high: Int
    let low: Int
    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)

            Text("\(high)")

            Text("\(low)")

        }
        .padding()
    }
}
```

Paso 5

Actualice sus inicializadores de vista de texto para incluir las etiquetas altas y bajas antes de las temperaturas.

Experimento

Intenta cambiar el texto de la cadena que rodea la interpolación de la temperatura. Por ejemplo, puedes escribir un signo de grado (°) usando la Opción-0.

Utilice propiedades calculadas para el icono y el color

Añade otra propiedad que utilice un booleano, un valor que sea verdadero o falso, para calcular lo que aparece en la vista de la imagen. Muestra una nube de lluvia si el valor es verdadero y un sol si es falso.

Paso 1

Añade una nueva propiedad booleana a DayForecast para representar las condiciones de lluvia.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)
            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)
        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String

    let isRainy: Bool

    let high: Int
    let low: Int

    var body: some View {
        VStack {
            Text(day)
            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)
            Text("\(high)")
            Text("\(low)")
        }
        .padding()
    }
}
```

Paso 3

Debido a que la propiedad `isRainy` es un `Bool`, necesitas escribir código para saber qué hacer cuando el valor es verdadero y cuando es falso. Para ello, crea una nueva propiedad calculada que represente el nombre del icono.

Una propiedad calculada no almacena un valor directamente como las propiedades almacenadas que declaró anteriormente usando `let`. Las propiedades calculadas utilizan la palabra clave `var` porque su valor puede variar dependiendo de los resultados del cálculo.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {

    }

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)

            Text("\(high)")

            Text("\(low)")

        }
        .padding()
    }
}
```

Paso 4

Examine el error en su nueva propiedad calculada.

El código que calcula un valor debe proporcionar el valor utilizando la palabra clave return. (Accessor es otro nombre para una propiedad calculada).

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int
    var iconName: String {

        return "cloud.raid.fill"

    }

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: "sun.max.fill")
                .foregroundColor(Color.yellow)

            Text("\(high)")

            Text("\(low)")

        }
        .padding()
    }
}
```


Paso 6

Reemplace el literal de cadena "sun.max.fill" en el inicializador de imágenes con una referencia a su nueva propiedad calculada iconName.

Ahora tienes dos nubes de lluvia amarillas. Necesitas usar el valor de isRainy para devolver diferentes valores.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int
    var iconName: String {
        return "cloud.raid.fill"
    }

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: iconName)

                .foregroundColor(Color.yellow)

            Text("\(high)")

            Text("\(low)")

        }
        .padding()
    }
}
```

Paso 7

Elimine la instrucción de retorno única en la propiedad calculada, luego agregue una instrucción if/else para devolver el nombre de icono correcto basado en el valor de la propiedad isRainy.

Si la condición es verdadera, Swift ejecuta el código en las llaves después del if. Si la condición es falsa, Swift ejecuta el código en las llaves después de la palabra clave else.

Propina

Al igual que la simple declaración de devolución, puede optar por eliminar las palabras clave de retorno en la declaración if/else.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {

    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {

        if isRainy {

            return "cloud.rain.fill"

        } else {

            return "sun.max.fill"

        }

    }

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: iconName)

                .foregroundColor(Color.yellow)

            Text("\(high)")

            Text("\(low)")

        }
        .padding()
    }
}
```

Paso 8

El color del icono también depende de la propiedad `isRainy`, por lo que puede seguir el mismo patrón para crear una segunda propiedad calculada para el color del icono. Utilice otra declaración `if/else` para devolver el color correcto dependiendo de las condiciones climáticas. A continuación, reemplace el valor `Color.yellow` en el modificador `.foregroundColor` con una referencia a `iconColor`.

Has encapsulado toda la expresividad de una tarjeta de pronóstico en una vista personalizada. Añade una o dos tarjetas más para ver lo fácil que es hacer otras nuevas.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {
        if isRainy {
            return "cloud.rain.fill"
        } else {
            return "sun.max.fill"
        }
    }

    var iconColor: Color {
        if isRainy {
            return Color.blue
        } else {
            return Color.yellow
        }
    }

    var body: some View {
        VStack {

            Text(day)

            Image(systemName: iconName)

                .foregroundColor(iconColor)

            Text("High: \(high)")

            Text("Low \(low)")

        }
        .padding()
    }
}
```

Dale estilo a la vista personalizada

Ahora que ha hecho una subvista personalizada para una tarjeta de pronóstico, puede ajustar fácilmente el estilo, la fuente y el relleno de sus elementos y ver los cambios en cada tarjeta.

Paso 1

En el cuerpo de su vista DayForecast, agregue una línea debajo de Texto (día). Luego, comience a escribir `.font` y haga una pausa para que Xcode muestre sugerencias de finalización del código.

```
M font(_ font:) >
M fontWeight(_ weight:) >
M fontWidth(_ width:) >
M fontDesign(_ design:) >
```

```
font(_ font: Font?) -> Text
Sets the default font for text in the view.
```

Paso 2

Elija `font(_ font:)` de las sugerencias de finalización del código y pulse Retorno para insertarlo en su código. A continuación, escriba para reemplazar el marcador de posición resaltado con `Font.headline`.

Algunas sugerencias de finalización de código, como el título y el título, son valores semánticos: sus nombres indican cómo se deben usar las fuentes en su aplicación.

Experimento

Intenta usar diferentes valores para el modificador `.font`.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {

            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)

        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {
        if isRainy {
            return "cloud.rain.fill"
        } else {
            return "sun.max.fill"
        }
    }

    var iconColor: Color {
        if isRainy {
            return Color.blue
        } else {
            return Color.yellow
        }
    }

    var body: some View {
        VStack {

            Text(day)
                .font(Font.headline)
            Image(systemName: iconName)
                .foregroundColor(iconColor)

            Text("High: \(high)")

            Text("Low: \(low)")

        }
        .padding()
    }
}
```

Paso 3

Usa `.font(Font.largeTitle)` para aumentar el tamaño de la imagen.

Debido a que los símbolos SF se utilizan a menudo junto con el texto, puede ajustar su tamaño utilizando nombres de fuentes semánticas. Están diseñados para trabajar en muchos tamaños y pesos diferentes.

Nota

Los símbolos SF funcionan automáticamente con [Dynamic Type](#), por lo que se escalarán adecuadamente cuando las personas ajusten el tamaño del texto en su dispositivo.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)
        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {
        if isRainy {
            return "cloud.rain.fill"
        } else {
            return "sun.max.fill"
        }
    }

    var iconColor: Color {
        if isRainy {
            return Color.blue
        } else {
            return Color.yellow
        }
    }

    var body: some View {
        VStack {

            Text(day)
                .font(Font.headline)
            Image(systemName: iconName)
                .foregroundColor(iconColor)
                .font(Font.largeTitle)

            Text("High: \(high)")

            Text("Low: \(low)")
        }
        .padding()
    }
}
```

Paso 4

Utilice el modificador `.fontWeight` para cambiar el peso de las temperaturas altas y bajas. Experimenta con diferentes pesos de las sugerencias de finalización del código.

El modificador `.fontWeight` cambia el peso de una fuente sin afectar su significado semántico (la forma en que lo haría el uso de `.body` o `.headline`).

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)
        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {
        if isRainy {
            return "cloud.rain.fill"
        } else {
            return "sun.max.fill"
        }
    }

    var iconColor: Color {
        if isRainy {
            return Color.blue
        } else {
            return Color.yellow
        }
    }

    var body: some View {
        VStack {

            Text(day)
                .font(Font.headline)
            Image(systemName: iconName)
                .foregroundColor(iconColor)
                .font(Font.largeTitle)

            Text("High: \(high)")
                .font(Font.Weight.semibold)

            Text("Low: \(low)")
                .font(Font.Weight.medium)

        }
        .padding()
    }
}
```

Paso 5

Usa el color del sistema secundario para la baja temperatura.

Utilizas el color secundario para obtener información menos importante.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)
        }
    }
}

#Preview {
    ContentView()
}

struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {
        if isRainy {
            return "cloud.rain.fill"
        } else {
            return "sun.max.fill"
        }
    }

    var iconColor: Color {
        if isRainy {
            return Color.blue
        } else {
            return Color.yellow
        }
    }

    var body: some View {
        VStack {

            Text(day)
                .font(Font.headline)
            Image(systemName: iconName)
                .foregroundColor(iconColor)
                .font(Font.largeTitle)

            Text("High: \(high)")
                .font(Font.Weight.semibold)

            Text("Low: \(low)")
                .font(Font.Weight.medium)

            .foregroundColor(Color.secondary)
        }
        .padding()
    }
}
```


Paso 6

Por último, añade un poco de relleno al icono.

Experimento

Añade tu propio estilo al proyecto, editando el aspecto y los colores hasta que estés satisfecho.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        HStack {
            DayForecast(day: "Mon", isRainy: false, high: 70, low: 50)

            DayForecast(day: "Tue", isRainy: true, high: 60, low: 40)
        }
    }
}

#Preview {
    ContentView()
}
```

```
struct DayForecast: View {
    let day: String
    let isRainy: Bool
    let high: Int
    let low: Int

    var iconName: String {
        if isRainy {
            return "cloud.rain.fill"
        } else {
            return "sun.max.fill"
        }
    }



    var iconColor: Color {
        if isRainy {
            return Color.blue
        } else {
            return Color.yellow
        }
    }

    var body: some View {
        VStack {

            Text(day)
                .font(Font.headline)
            Image(systemName: iconName)
                .foregroundColor(iconColor)
                .font(Font.largeTitle)
                .padding(5)

            Text("High: \(high)")
                .font(Font.Weight.semibold)

            Text("Low: \(low)")
                .font(Font.Weight.medium)
                .foregroundColor(Color.secondary)
        }
        .padding()
    }
}
```

Mon	Tue
	
High: 70 Low: 50	High: 60 Low: 40