# LÍNEAS DE PRODUCTOS SOFTWARE

---

## Sesión 03. XCode, Swift, iOS: Vistas personalizadas y modelo de datos

Aitor García Luiz
*Grado en Ingeniería Informática*

La sesión 03 de prácticas parte de lo realizado anteriormente en la sesión 02. Antes de comenzar los ejercicios se deberá extender la misma siguiendo los pasos del guion de prácticas.

De esta forma, se ha añadido una nueva zona debajo del área de la imagen, en la que hasta el momento se han colocado cinco botones cuadrados cuya funcionalidad es imprimir una misma frase en la consola.

## Ejercicio 1.

**Vamos a modificar el código que sea necesario para que cada botón muestre un mensaje diferente. Como único método de acción tendremos a btnEval(:_), aunque ahora mostrará el mensaje "Botón x pulsado", donde x será el número de botón correspondiente.**

**Como solución rápida, investigad el uso del atributo tag:**

La nueva función de btnEval() quedaría de la siguiente forma:

```
func btnEval(boton: UIButton) {
    print("Botón \(boton.tag) pulsado...👍")
}
```

Mostrando las siguientes frases en la consola:

```
Botón 4 pulsado...👍
Botón 5 pulsado...👍
Botón 3 pulsado...👍
Botón 1 pulsado...👍
```

Haciendo uso del atributo tag, cuando se crean los botones se les ha asignado un tag de 1 a 5 para identificarlos posteriormente, y en esta función podemos hacer uso de ellos para saber que botón es el que ha mostrado la frase en la consola.

## Ejercicio 2.

**Vamos a parametrizar el código, haciendo uso de constantes en lugar de utilizar literales (calidad). El tamaño del botón estará determinado por el valor del frame que lo contiene, y que ha sido establecido en tiempo de compilación.**

Para realizar este ejercicio nos centraremos especialmente en la función que inicializa los botones. Dentro de esta teníamos la siguiente línea de código:

```
let boton = UIButton(frame: CGRect(x: 0, y: 0, width: 44, height: 44))
```

La cual será modificada de la siguiente forma:

```
let boton = UIButton(frame: CGRect(x: 0, y: 0, width: self.frame.width
                   / 5, height: self.frame.height))
```

De esta forma los botones variarán en función del tamaño del frame que los contiene.

## Ejercicio 3.

**Vamos a investigar cuántos observadores podemos crear para cada uno de los atributos. Especificad y justificar el comportamiento del que consideréis más interesante.**

Se pueden crear dos tipos diferentes de observadores, willSet y didSet. Ya hemos visto didSet en el guión de prácticas, por lo que sería interesante conocer el funcionamiento del observador willSet.

El observador willSet funciona de manera similar a didSet con la gran diferencia que willSet es invocado antes de que el valor se cambie y no después. El parámetro que se pase al observador willSet es una constante que tiene el nuevo valor de la propiedad. Además, habría que tener en cuenta que ningún observador de propiedades invocado cuando se inicia la instancia.

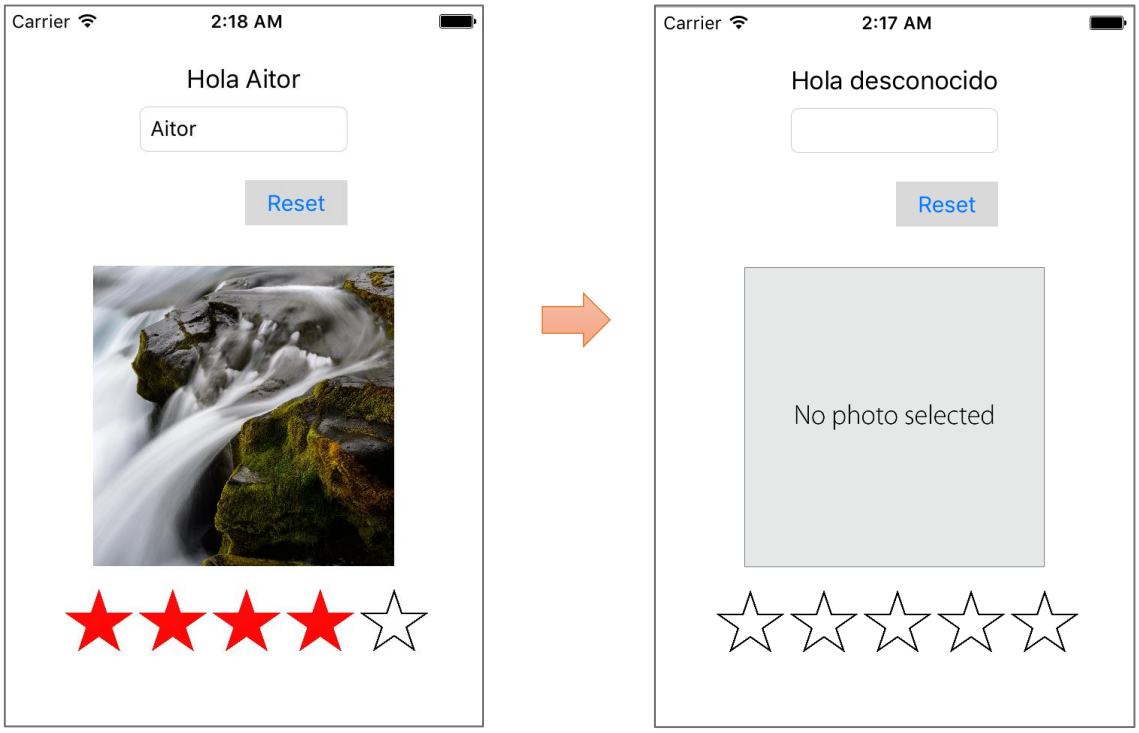
## Ejercicio 4.

**Vamos a ampliar la funcionalidad del botón Reset para dejar a 0 el valor gradoAfinidad… con su consecuente actualización de la vista.**

Ampliar la funcionalidad del botón reset es bastante simple, solamente deberemos ir a la función de dicho botón (resetBT() ) y añadir la siguiente línea:

```
controlEvaluacion.gradoAfinidad = 0
```

De esta forma, actualizará el estado del grado de afinidad poniéndolo a 0 lo que limpiará las estrellas que ya habían sido seleccionadas:



## Ejercicio 5.

**¿Qué habría pasado si, en lugar de hacer uso del método XCTAssertNil hubiésemos especificado XCTAssertNotNil? Documente la respuesta.**

Si hubiésemos especificado XCTAssertNotNil en lugar de XCTAssertNil, los tests no se hubieran pasado correctamente a menos que se editasen.

Por ejemplo, si nos fijamos en la constante 'posibleAmigo2', en ella se crea un nuevo Amigo cuyo nombre es una cadena de texto vacía y según la sentencia condicional que hemos añadido en el constructor de dicho Amigo, es necesario introducir un nombre, por lo que devolverá el valor 'nil' al no tenerlo. Si volvemos al test y continuamos, gracias al método XCTAssertNil, se comprueba que el valor devuelto sea nil, que como hemos comentado así es, por lo que comprobará que ambos valores son iguales y si pasará el test. Si por el contrario usáramos XCTAssertNotNil, fallaría el test al comprobar que el valor devuelto que hemos obtenido (nil) no sea nil.