1

SWIFT

Swift es un nuevo lenguaje de programación para iOS, OS X, watchOS y aplicaciones tvOS basado en lo mejor de C y Objective-C, sin las limitaciones de compatibilidad de C. Añade patrones de programación segura y características modernas para que la programación sea más fácil, flexible y divertida. Con Swift han querido hacer borrón y cuenta nueva para poder re-imaginar el desarrollo de software.

Swift resulta familiar para los desarrolladores de Objective-C, no ha cambiado tanto la forma en que se desarrollan las aplicaciones, lo cual facilita el cambio del nuevo lenguaje. Adopta la legibilidad de los patrones con nombre de Objective-C y el poder del modelo de objetos dinámicos, esto posibilita combinar Swift con el código Objective-C. Con Swift se introducen muchas nuevas características y se unifican las partes procesales y de orientado a objetos del lenguaje.

Para los nuevos desarrolladores, Swift es muy amigable. Es el primer lenguaje de programación para software de altas prestaciones tan expresivo y agradable como un lenguaje de *script* sencillo. Es compatible con el Playground, una característica innovadora que permite a los programadores experimentar con el código y ver los resultados inmediatamente sin la sobrecarga del desarrollo de una aplicación.

En mi opinión, Swift ha conseguido ser un lenguaje de programación moderno, preparado para las nuevas generaciones y muy amigable, que consigue que nos centremos en la lógica de la aplicación y no en el lenguaje en sí. Es como disfrutar de conducir sin pensar en cómo se conduce el coche. Todo esto mezclado con la sabiduría de la cultura más amplia de la ingeniería Apple.

El compilador está optimizado para el rendimiento y el lenguaje para el desarrollo, sin comprometer a ambos. Por eso Swift es una inversión de futuro para todos los desarrolladores que queramos vivir en el mundo de Apple.

Tenemos que pensar que Swift es un "recién nacido", pero con la experiencia del más veterano. Por ello, nos vamos a encontrar con un lenguaje en constante evolución en las nuevas características y capacidades.

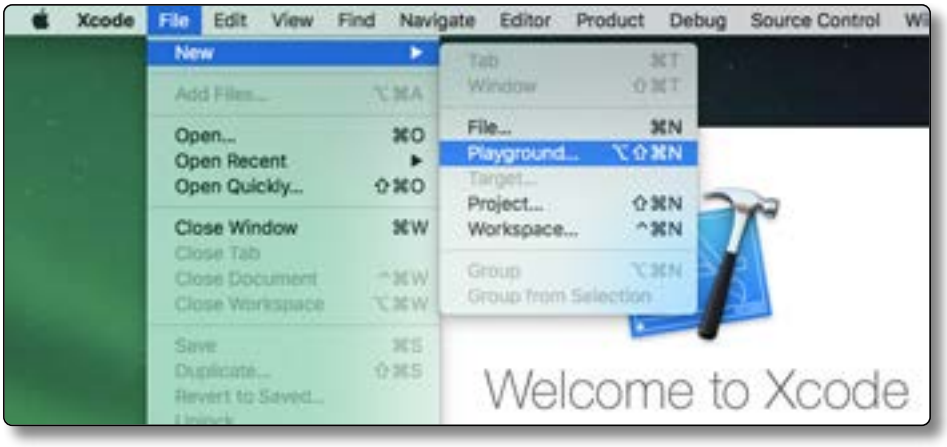
1.1 PLAYGROUND

Para empezar a conocer el nuevo lenguaje de Apple, Swift, tenemos que conocer "el parque infantil".

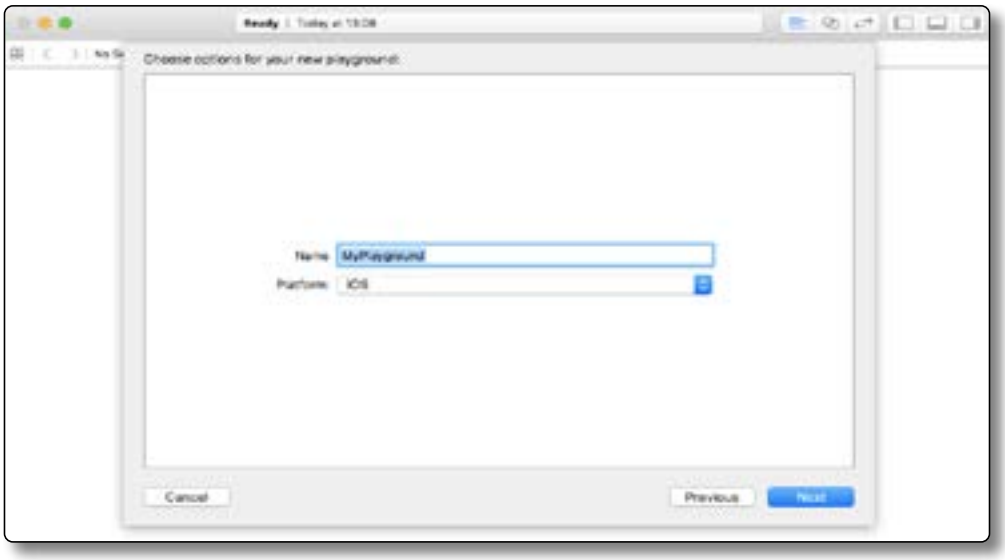
El Playground (parque infantil) es un entorno de codificación Swift interactivo que evalúa cada declaración y muestra los resultados cuando se realizan cambios sin la necesidad de crear un proyecto. Esto facilita mucho el aprendizaje del lenguaje, así como la puesta en práctica de pequeños algoritmos que quieras desarrollar para tu proyecto real. Se utiliza para aprender y explorar Swift, como piezas prototipo para una aplicación y para crear ambientes de aprendizaje. El entorno interactivo de Swift te permite experimentar con algoritmos, explorar las API del sistema e incluso crear vistas personalizadas.

Es, por todo esto, un entorno perfecto para empezar a experimentar con este fascinante lenguaje. ¡Manos a la obra!

En primer lugar vamos a ejecutar Xcode, donde nos saldrá nuestra ventana de bienvenida. De todas las opciones seleccionaremos **Get started with Playground**. En caso de que no aparezca la ventana de bienvenida podremos crear un nuevo Playground accediendo al menú de herramientas de Xcode, **File → New → Playground**.



A continuación nos saldrá una ventana donde introducir el nombre de nuestro fichero Playground. En **Plataforma** podemos seleccionar para qué sistema operativo queremos experimentar, en nuestro caso iOS:



Presionamos sobre el botón **Next**, indicamos una ubicación para guardar el fichero y ya tenemos nuestro Playground preparado para estudiar:

El Playground es como un bloc de notas moderno muy sencillo de usar. Tenemos en la parte de la izquierda, la pizarra, donde insertar nuestro código; en la derecha vemos línea por línea el resultado de dicho código cada vez que detecte un cambio.

Al iniciar un nuevo Playground nos encontramos con el siguiente código:

```
//: Playground - noun: a place where people can play

import UIKit

var str = "Hello, playground"
```

Tenemos un comentario con las dos barras //, sentencia de código que el compilador no interpreta. A continuación tenemos un **import UIKit**, esto añade a nuestro Playground un conjunto de funciones con respecto a la interfaz del usuario. Esta librería nos permitirá experimentar con elementos de la interfaz del usuario y aprender un poco más de ellas.

Finalmente nos encontramos con la definición de una variable llamada **str** con el valor **Hello, playground**.

Todos estos elementos los estudiaremos más adelante. De momento lo importante es notar que, junto a la línea de la variable, podemos ver, en la derecha de la pantalla, su contenido.

Con todo esto, vamos a empezar a ver los conceptos básicos del lenguaje.

1.2 BÁSICOS

Swift proporciona sus propias versiones de todos los tipos C y de Objective-C fundamentales, incluyendo **Int** para enteros, **doubles** y **flotas** para valores en punto flotante, **Bool** para valores booleanos y **String** para los datos textuales.

Al igual que C, Swift utiliza variables para almacenar y hacer referencia a valores con un nombre de identificación; así como las constantes, variables cuyos valores no se pueden cambiar, que son mucho más potentes que las constantes en C.

Swift introduce tipos avanzados que no se encuentran en Objective-C, como las tuplas, elementos que te permiten crear y pasar una agrupación de valores. Se pueden usar para devolver varios valores de una función como si fueran un único valor.

Una parte un poco más complicada de entender son los valores opcionales, algo novedoso en Swift; son comparables a los valores nulos en Objective-C. Son los valores encargados de ocupar la ausencia de un valor en una variable no solo de una clase sino de cualquier tipo. Son más seguros que los punteros nulos de Objective-C.

1.2.1 Comentarios

Usamos los comentarios para incluir texto no ejecutable en el código, como una nota o recordatorio. Los comentarios son ignorados por el compilador Swift cuando se compila el código.

Los comentarios en Swift son muy similares a los de cualquier otro lenguaje de programación, para emplearlos hay que introducir la doble barra inclinada al principio de una línea (//) y comentar la sentencia. En caso de más de una línea utilizamos /* al principio de la sentencia y terminamos con */.

```
//Experimentando con el lenguaje Swift

/*
    Comentarios con muchas líneas
    Podemos escribir un texto explicativo o recordatorio
*/
```

1.2.2 Constantes y variables

Las constantes y variables son espacios en memoria de la máquina con un nombre asociado a un valor de un tipo particular. Las constantes son variables que no pueden cambiar su valor una vez está establecido, mientras que en una variable se puede establecer un valor diferente en cualquier momento.

Para poder usar las constantes y variables, antes se tienen que declarar, proceso muy sencillo en Swift. Para declarar una variable constante tenemos que usar la palabra reservada **let** seguida del nombre de la constante y un valor. Para el caso de las variables se utiliza la palabra reservada **var**. Vamos a ver un ejemplo. Introducimos en nuestro Playground las siguientes sentencias de código:

El código se lee de la siguiente forma:

```
var numDeMascotas = 2 //variable
let numPI = 3.1416 //constante
```

Hemos creado una variable llamada **numDeMascotas** con el valor asociado 2. Después hemos declarado una constante llamada **numPI** con el valor 3.1416.

Antes de seguir vamos a prestar atención a varias cosas en la declaración de las variables.

1. Los nombres que pongamos a nuestras variables deben empezar por minúsculas sin acentos y sin espacios en blanco.

2. No hace falta terminar la sentencia con un punto y coma.

3. La notación correcta para definir valores es con un símbolo igual (=).

4. Los números se introducen sin comillas. El punto es el separador de los decimales.

5. Las cadenas o valores alfanuméricos han de ir entre comillas dobles; si queremos definir un carácter, podemos poner el valor entre comillas simples.

También podemos declarar múltiples constantes y variables en una línea, separadas por comillas:

```
var x = 29.0, y = 1.72, z = 65.7
```

En Swift se hace mucho hincapié en usar constantes si el valor de la variable no va a modificar su valor. Las variables se utilizan solo para almacenar valores que deben ser capaces de cambiar. A lo largo de los proyectos, el compilador marcará aquellas variables que no cambian en su valor, sugiriendo que lo modifiques con la declaración **let**, constante. Ya que de esta forma optimizamos mucho mejor nuestras aplicaciones en cuanto a memoria y rendimiento.

La función print()

Finalmente, para poder pintar una variable o constante utilizamos la función **print(nombreVariable)**:

```
print(numDeMascotas) //pintando 2
```

La función **print** permite imprimir uno o varios valores por la consola del compilador. Por defecto, la función termina la línea que se imprime con un salto de línea (**/n**).

Con Swift podemos usar interpolación de cadenas para incluir el nombre de una constante o variable como un marcador de posición de una cadena más larga envolviendo la variable o constante entre paréntesis y con una barra invertida al principio:

```
print("El número de mascotas es: \(numDeMascotas)")
```

Esta sentencia pintará “El número de mascotas es: 2”.

1.2.3 Tipos de anotaciones

En la declaración de las variables anteriores podemos observar que no hemos indicado el tipo de variable que podemos almacenar. Esto nos facilita a la hora de declarar una variable de la que no sabemos el tipo de valor, pero en ocasiones necesitamos anotar qué tipo de valor puede almacenar para conseguir una seguridad en nuestro programa. En Swift, como en otros lenguajes de programación, tenemos lo **String**, **int** y **dobule**, **Boolean** para los diferentes valores de una variable.

Para poder definir una variable o constante indicando el tipo se introducen dos puntos después del nombre seguido de un espacio en blanco y del nombre del tipo que se va a usar.

Vamos a declarar una variable llamada **nombre** que puede almacenar valores de tipo **String**, es decir, un valor textual:

```
var nombre : String
```

Ahora la variable nombre puede contener cualquier valor de tipo cadena sin error:

```
nombre = "Enrique"
```

También podemos definir múltiples variables del mismo tipo en una línea de código, separados por comas con el tipo de anotación al final de los nombres de las variables:

```
var peso, altura: Double
```

1.2.4 Números enteros

Las variables y constantes de tipo enteros pueden almacenar valores numéricos no decimales positivos, cero o negativos.

```
let min = UInt8.min //igual a 0
let max = UInt8.max //igual a 255
```

Con Swift podemos asignar números enteros de 8, 16, 32 y 64 bits.

En muchas ocasiones no queremos especificar el tamaño del número entero. Swift ofrece un tipo entero adicional, el **Int**:

```
let numTransportes: Int = 6765
```

64 bits obtiene el tamaño de un **Int64**. A menos que se necesite trabajar con un tamaño específico de número entero, se recomienda utilizar el **Int** para valores enteros. Esto ayuda a la consistencia del código, incluso en plataformas de 32 bits. **Int** puede almacenar cualquier valor comprendido entre -2147483648 y 2147483647, un rango lo suficientemente grande para almacenar una gran cantidad de números enteros.

1.2.5 Números de punto flotante

Los números de punto flotante son números con decimales, positivos y negativos.

Los tipos de punto flotante pueden representar una gama mucho más amplia de valores que los tipos enteros. En Swift existen dos tipos de número con punto flotante:

1. **Double**: Representa un número de 64 bits con una precisión de unos 15 dígitos decimales.

2. **Float**: Representa un número de 32 bits con una precisión de unos 6 dígitos decimales.

El tipo de punto flotante apropiado depende de la naturaleza y el rango de valores que necesita para trabajar en el código. En situaciones en las que cualquiera de los tipos sería apropiado se prefiere el **Double**:

```
let peso: Double = 65.5678
let altura: Float = 1.765
```

1.2.6 Booleans

En Swift existe un tipo booleano básico llamado **Bool** cuyo valor puede contener **true** o **false**:

```
var elCieloEsAzul : Bool = true
var elSolEsAmarillo : Bool = false
```

Los valores booleanos son particularmente útiles cuando se trabaja con sentencias condicionales tales como la sentencia **if**:

```
if elCieloEsAzul {
    print("El cielo es azul")
```

```
}else{
    print("El cielo NO es azul")
}
```

1.2.7 Tuplas

Las tuplas permiten contener múltiples valores en un único valor compuesto. Los valores dentro de una tupla pueden ser de cualquier tipo y diferentes entre sí:

```
var http404Error = (404,"Not Found")
```

En este ejemplo vemos una tupla con un entero y una cadena de texto separados con una coma y entre paréntesis. Se pueden crear tuplas de cualquier permutación de tipos, y pueden contener tantos tipos diferentes como se desee.

A la hora de declarar una tupla, no solo ha creado nuestra variable con valores múltiples, sino que les ha asignado un índice a cada uno de manera automática empezando por la posición 0.

En este caso ha asignado el valor 0 para el entero y el valor 1 para el texto, es decir, en caso de que queramos pintar los valores de la tupla, introduciremos lo siguiente:

```
print(http404Error.0) //pinta 404
print(http404Error.1) //pinta Not Found
```

Si queremos cambiar un valor, solo tenemos que asignar un nuevo valor al índice directamente:

```
http404Error.1 = "Página no encontrada"
print(http404Error.1)
```

Debemos tener cuidado en no acceder a valores de índice que no existen, ya que provocaría un error.

El uso de índices puede no ser muy fácil de recordar por parte del programador. En caso de tener muchos elementos dentro de la tupla podemos olvidar qué posición ocupa un determinado valor. Para solucionar esto, Swift proporciona la capacidad de nombrar los elementos de la tupla para no tener que recordar el orden en que fueron puestos a la hora de declarar:

```
var http404Error = (error: 404,mensaje: "Not Found")
```

Ahora hemos puesto un nombre a cada uno de los valores de la tupla. Internamente para Swift siguen teniendo un índice por posición (0,1) pero ahora el

programador puede acceder al valor de los elementos introduciendo el nombre; así, el código es más fácil e intuitivo:

1.2.8 Opcionales

Los opcionales se utilizan en situaciones en las que un valor puede estar ausente.

Este concepto no existe en C u Objective-C. Lo más cercano en Objective-C es **nil**, que significa la ausencia de un objeto válido. Sin embargo, esto solo funciona para objetos y no para tipos básicos o valores de enumeración. Para estos tipos, normalmente, devuelven en Objective-C un valor especial (**NSNotFound**) para indicar la ausencia de un valor. Los valores opcionales de Swift permiten indicar la ausencia de un valor de cualquier tipo sin la necesidad de constantes especiales.

Vamos a ver un ejemplo de cómo usar opcionales para hacer frente a la falta de un valor:

```
let numString = "123"
let numInt = Int(numString)
```

Vemos que tenemos una variable **numInt** en la cual estamos intentando transformar un tipo **String** en un **Int.** Esto es un poco peligroso, ya que no todas las cadenas se pueden convertir en un número entero. En este caso se puede, pero con una cadena como, por ejemplo, "Hola mundo" no podríamos.

Debido a que la conversión puede fallar, la función **Int(numString)** devuelve un **Int** opcional, en lugar de un **Int.**, lo que significa que puede contener un valor **int** o ninguno. Se evitan así posibles errores:

```
var variableOpcional : Int?

variableOpcional = 123
variableOpcional = nil
```

Para declarar una variable que pueda contener un valor opcional se pone el símbolo de cierre de interrogación (**?**) después del tipo de variable. En el ejemplo anterior no tenemos un tipo **Int**, sino **Int?**

Podemos observar que la variable opcional puede contener el valor entero o un valor **nil** (un valor vacío). En caso de declarar un variable opcional y no asignar un valor inicial, Swift asigna el valor **nil** de forma automática.

Ahora vamos a pintar el valor de una variable opcional:

Podemos observar que al pintar una variable opcional no se pinta el valor, sino que se introduce la palabra **Optional(123)** con el valor en el interior de los paréntesis. Para poder acceder al valor real de una variable opcional tenemos que utilizar el símbolo de cierre de exclamación (**!**) al final de la variable. De esta forma forzamos el desempaquetar el valor de una opcional:

Ahora sí que vemos el valor real de la variable.

El forzar una variable opcional para obtener un valor que contenga **nil** puede provocar un error en tiempo de ejecución, por eso se recomienda comprobar que el valor es distinto a vacío antes de forzar el obtener el valor.

Llegados a este punto ya tenemos los conceptos básicos en el lenguaje Swift:

1. Comentarios
2. Variables y constantes
3. Tipo de variables y declaración
4. Función **print**
5. Números enteros
6. Números de punto flotante
7. **Booleans**
8. Tuplas
9. Opcionales

El siguiente paso son los operadores y expresiones. ¡Tómate un descanso antes de seguir!

1.3 OPERADORES Y EXPRESIONES

Los operadores en Swift son los mismos que en cualquier otro lenguaje, el símbolo que se utiliza para comprobar, cambiar o combinar valores.

Existen tres tipos diferentes de operadores:

1. **Unarios**: aquellos operadores que operan en un solo valor. Los unarios prefijo aparecen inmediatamente antes de su destino (por ejemplo, **!a**); luego tenemos los unarios postfijos sencillos, aquellos que aparecen inmediatamente después de su objetivo (por ejemplo el incremento en uno, **a++**).

2. **Binarios**: los operadores más comunes, ya que operan con dos valores, por ejemplo, **a + b**.

3. **Ternarios**: operadores que operan en tres valores. En Swift existe un único operador ternario, el operador condicional **a ? b: c**

1.3.1 Operador de asignación

El operador de asignación es aquel que inicializa o actualiza un valor **b** con el de **a (b = a)**.

```
let a = 20                                    20
var b = 4                                     4

b = a                                         20
```

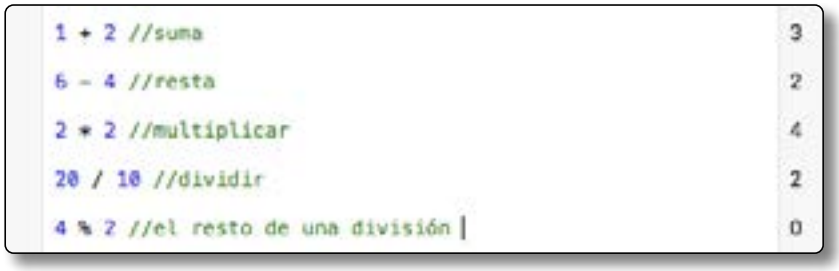
A diferencia del operador de asignación de C y Objective-C, en Swift no devuelve ningún valor, de esta forma no lo confundimos por error con el operador de comparación (==):

```
if a = b {
    //error al usar un operador de asignación y no de
        comparador
}
```

1.3.2 Operadores aritméticos

Los operadores aritméticos son los más comunes y fáciles de usar.

1. Sumar (+)
2. Restar (-)
3. Multiplicar (*****)
4. Dividir (/)
5. Resto (**%**)



1.3.3 Operadores de incremento y decremento

Los operadores de incremento y decremento son operadores unarios que aumentan (++) o disminuyen (--) el valor de una variable numérica en uno:

Cada vez que se llama al operador, el valor de la variable se incrementa o disminuye en uno. Los operadores de incremento y decremento son atajos para decir **i = i +1** y **i = i-1**, respectivamente.

Debemos tener en cuenta que estos operadores modifican el valor de la variable en la cual operan y podemos colocar el operador delante o detrás de la variable:

1. **Antes de la variable**: se incrementa la variable antes de devolver su valor.

2. **Después de la variable**: se incrementa la variable después de obtener su valor original.

1.3.4 Operadores de asignación compuestos

Los operadores de asignación compuestos son aquellos que combinan asignación (=) con otra operación:

La operación **a + = 2** es una abreviatura de **a = a + 2**. Lo único que debemos tener en cuenta es que las operaciones de asignación compuestas no devuelven un valor, por lo que no podemos asignar a una variable la operación compuesta **b = a+=2**.

1.3.5 Operadores de comparación

Los operadores de comparación son aquellos que comparan dos valores y devuelven un valor booleano en caso de que se cumpla:

Los operadores por sí solos no tienen mucho sentido, por lo que se suelen usar sentencias de condición:

1.3.6 Operadores lógicos

Los operadores lógicos son aquellos que necesitan de la combinación de dos valores lógicos booleanos **true** y **false**, y devuelven un valor booleano resultante de la operación. En Swift existen los tres operadores lógicos estándar que se encuentran en lenguajes basados en C:

1. **NOT** lógico (**! a**)
2. **AND** lógico (**a && b**)
3. **OR** lógico (**a || b**)

Al igual que ocurre con los operadores de comparación, los operadores lógicos se suelen usar bajo una sentencia que contenga un condicional **if**.

El operador lógico **NOT** invierte un valor booleano donde un valor verdadero se convierte en falso, y a la inversa:

El operador lógico **AND** genera un verdadero cuando ambos valores son verdaderos en la expresión. En otros casos siempre es falso:

```
let enteredDoorCode = true                                  true
let passedRetinaScan = false                                false
if enteredDoorCode && passedRetinaScan {
    print("Welcome!")
} else {
    print("ACCESS DENIED")                                  "ACCESS DENIED\n"
}
```

El operador lógico **OR** se utiliza para crear expresiones lógicas en las que solo uno de los valores tiene que ser verdadero para que genere un valor verdadero. En caso de ser los dos falsos devolverá un valor falso:

```
let hasDoorKey = false                                      false
let knowsOverridePassword = true                            true
if hasDoorKey || knowsOverridePassword {
    print("Welcome!")                                       "Welcome!\n"
} else {
    print("ACCESS DENIED")
}
```

Se pueden combinar múltiples operadores lógicos para crear expresiones compuestas más largas:

```
if enteredDoorCode && passedRetinaScan || hasDoorKey ||
    knowsOverridePassword {
    print("Welcome!")                                       "Welcome!\n"
} else {
    print("ACCESS DENIED")
}
```

En resumen, los operadores en Swift son los siguientes:

1. De asignación (=)
2. De comparación (==, >, <, >=, <=, **!**=)
3. Aritméticos (+, -, *, /, **%**)
4. De asignación compuestos (+=, -=, *=, /=)
5. De incremente y decremento (++, —)
6. Lógicos (**&&**, ||)