PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Introducción al lenguaje Swift
Variables
Constantes
Operadores
Arrays

Introducción a Swift

Vamos a aprender las bases del lenguaje de programación usado para el desarrollo de aplicaciones móviles para dispositivos con **sistema operativo iOS**: **Swift**.

Ya sabemos que para desarrollar aplicaciones para dispositivos Android, se pueden utilizar, entre otros, dos lenguajes: Java y Kotlin. Algo similar ocurre para el desarrollo de aplicaciones para **dispositivos iOS**. En estos, podremos utilizar los lenguajes Objective-C y Swift. **En un principio**, las aplicaciones se desarrollaban íntegramente en C#, pero **hoy en día**, se están migrando de una forma bastante rápida a **Swift**.

El lenguaje de programación Swift fue desarrollado por Apple y es:

- un lenguaje **multiparadigma**,
- soportando:
  - **orientación a objetos**,
  - **orientación a protocolos**,
  - **programación funcional**
  - y **programación imperativa**.

Mediante este lenguaje se podrán realizar aplicaciones tanto para dispositivos **móviles con iOS** como aplicaciones para el **sistema operativo macOS**.

Apple presentó este nuevo lenguaje de programación en el año **2014**, en la WWDC (Worldwide Developers Conference), que es como se denominan las **conferencias ofrecidas por Apple** para **presentar sus productos**.

A pesar de ser un lenguaje de programación diseñado por Apple, lo presentaron como ***software libre***, algo bastante contrario a la hermeticidad de esta compañía. Además, este lenguaje es multiplataforma, con lo cual, podremos **desarrollar** aplicaciones escritas **en** él en cualquiera de los tres grandes sistemas operativos: **GNU/Linux, Windows y,** por supuesto, **macOS**.

Instalación de Swift en Windows

Para instalar Swift en Windows, necesitamos descargar el programa **Swift for Windows**, el cual nos instalará el **compilador de Swift** para Windows y un programa para poder compilar y ejecutar los programas que desarrollemos.

Para descargarlo, nos haremos de la siguiente página:

https://swiftforwindows.github.io/

Seleccionamos Download y empezará la descarga.

Una vez descargado, lo instalaremos pulsando siguiente en todas las pantallas del asistente.

Una vez instalado, tendremos el **compilador listo**. Los programas que estén escritos en Swift deberán tener la extensión .swift. Desde Swift 2.0 podremos **compilar un programa** pulsando el botón Compile y seleccionando el fichero de código, y para poder **ejecutar en una terminal** el programa, pulsaremos el botón **Run** y seleccionaremos el fichero.

Para evitar llevar a cabo estos pasos, vamos a utilizar el programa Visual Studio Code, que integra una terminal en el propio editor y nos permitirá escribir el código, compilarlo y ejecutarlo de forma muy sencilla.

Recordamos que para descargar Visual Studio Code, lo podemos hacer desde su página web:

https://code.visualstudio.com/download

Una vez instalado y con un **fichero de código abierto**, podremos pulsar en el menú Terminal New Terminal y nos aparecerá una terminal integrada en la que podremos compilar nuestros programas, accediendo primero a la ubicación del compilador con el comando:

```
cd C:\Swift\bin
```

y, posteriormente:

- Para compilar: .\swiftc.exe ruta_fichero.swift
- Para ejecutar: .\swift.exe ruta_fichero.swift

Instalación de Swift en otros sistemas operativos

Para instalar el compilador de Swift en GNU/Linux tendremos que descargarlo de la siguiente página web https://swift.org/download y seleccionar la distribución deseada según nuestro sistema. Una vez descargado lo descomprimimos y lo copiamos en la carpeta /usr/, combinando todos los ficheros que nos indique.

En los sistemas operativos de MacOS ya viene instalado por defecto.

Dónde desarrollar con Swift

El lenguaje de programación Swift es un lenguaje multiplataforma, que, si recordamos, quiere decir que se podrá instalar en los tres grandes sistemas operativos del mercado: GNU/Linux, Windows y macOS.

Visto esto, podremos desarrollar programas escritos en Swift en cualquiera de estos sistemas operativos, pero esto no implica que se puedan desarrollar aplicaciones para iOS desde ellos. **El desarrollo de aplicaciones para iOS se podrá realizar única y exclusivamente desde el sistema operativo macOS**. Esto quiere decir que desde GNU/Linux y Windows podremos realizar programas escritos en Swift sin ningún inconveniente, pero **no podremos realizar aplicaciones para móviles iOS**.

Algo totalmente distinto ocurre con Android, ya que Android Studio se puede instalar en cualquiera de los tres grandes sistemas operativos.

Comentarios al código

Todos los lenguajes de programación deben de proporcionar un método para la implementación de comentarios en el código.

Ya sabemos que los comentarios nos ayudarán a introducir aclaraciones en el código, que nos facilitarán entender cómo están implementados los algoritmos que utilicemos para resolver cada problema en cuestión.

En los comentarios, recordemos que podremos escribir lo que necesitamos, incluso haciendo uso de cualquier símbolo o carácter que no podamos utilizar a la hora de programar, como pueden ser las tildes, la letra ñ, etc.

Swift no es una excepción y también proporciona un mecanismo de comentarios bastante simple y fácil de utilizar.

En primer lugar, tenemos los comentarios de una única línea, que nos ayudarán a escribir comentarios aclaratorios de una línea de longitud. Estos comentarios se implementan mediante **dos barras inclinadas**, y todo lo que se situé a partir de ellas, será considerado como comentario al código que le preceden.

```
// Este es un comentario de una sola linea con ñ y öóô
```

El otro tipo de comentario que nos permite implementar Swift es el comentario multilínea, en el cual, podremos escribir todas las líneas que necesitemos, considerándose como un único comentario. Estos comentarios podrán **escribirse entre /* y */**.

Comentario multilínea

```
/*
Esto es un ejemplo de
comentario de
varias lineas
*/
```

Como podemos observar, la implementación de los comentarios que nos proporciona Swift es idéntica a la que nos proporciona Java, por lo que no vamos a tener ninguna dificultad en adaptarnos a ella.

El color verde que tienen los comentarios en estos ejemplos es debido al uso de Visual Studio Code.

Variables y operaciones

El lenguaje de programación Swift nos ofrece un gran abanico de variables que podremos utilizar en nuestros programas, pero, al fin y al cabo, los tipos más complejos de datos que podremos utilizar van a estar basados en los tipos de datos básicos, como ocurría en los demás lenguajes de programación.

Los tipos de datos básicos que vamos a poder utilizar en Swift son los siguientes:

- Int: Este tipo de dato representa los números enteros con **32 bits** con un rango de entre menos 2.147.483.648 y mas 2.147.483.647.
- Double: Este tipo de datos representa los números reales con **64 bits** con hasta **15 decimales de** precisión.
- Bool: Este tipo de datos representa los datos booleanos que puede ser **0 (false**) o 1 (true).
- String: Este tipo de datos representa una cadena de caracteres.

En Swift, las variables, **al declararse**, **no llevarán indicado un tipo**, obteniéndolo en el momento en el que se igualan a algo por primera vez. Para declarar una variable, utilizaremos la palabra reservada **var**, mientras que para declarar una constante, utilizaremos la palabra reservada **let**. También **será posible indicar el tipo al declarar las variables**, aunque esto sería un poco inusual.

Ejemplo de declaración de variables en Swift

```
// Declaración de variables enteras
var numero1 = 8, numero2 = 3
// Declaración de variables de un tipo concreto
var palabra:String
palabra = “Hola”
// Declaración de una constante real
let PI = 3.141592
```

Las **operaciones** que podremos hacer con variables son las mismas que ya conocemos:

- Para datos del tipo numérico: Suma, resta, producto, división y **cálculo del resto**.
- Para datos del tipo cadena: **Concatenación** y demás operaciones que permita la clase.

Las variables en Swift **siempre deberán tener un valor**, es decir, no se podrán tener variables sin inicializar.

Variables opcionales

Swift no admite variables sin inicializar, sino que **se establecerá en el futuro**, pero **sí** que podemos **inicializar una variable sin saber qué valor** tendrá en ese momento,.

Esto se hace con un **'?' tras el tipo** de variable.

Con la interrogación, estamos indicando a Swift que la variable es de un tipo, pero que aún no queremos o no podemos asignarle un valor, por lo que se le **asignará automáticamente un valor nulo**.

Cuando queramos **recuperar dicho valor** tendremos que añadir el **modificador '!' (fin de exclamación)** para que extraiga de este contenedor opcional el valor, y lo **muestre tal cual**.

Entrada y salida de datos

Cualquier lenguaje de programación que se precie necesitará proveer algún mecanismo para poder leer información de teclado y mostrar información por pantalla.

Para **leer datos de teclado** en Swift, utilizamos la función readLine(), la cual nos **devolverá siempre un String**.

En el caso de que queramos leer un **entero o un real**, tendremos que hacerles un **casting** para convertir el String que hemos leído al dato que queramos.

Esta función nos va a **devolver una variable opcional**, por lo que, para usarla, deberemos **tratarla como tal**, y usar el o**perador '!' (fin de exclamación)** cuando la invoquemos.

Para poder imprimir por pantalla en Swift, vamos a utilizar la instrucción print, de la forma print ("Hola Mundo").

Otro elemento que podremos utilizar son los especificadores para indicar que, en ese lugar, irá el **contenido de una variable (parecido al f "{}"en python)**. En Swift, podemos hacerlo de la forma \(variable) (barra inclinada a izquierda seguido de la variable entre paréntesis). A esto lo conocemos como **interpolación de strings**.

Un aspecto que tenemos de tener en cuenta es que en Swift las **instrucciones no terminarán con un punto y coma** (;), al igual que ocurría en Java. No obstante, en el caso de que queramos anidar múltiples instrucciones **en una única línea** de código, sí que tendremos que **separarlas mediante punto y coma**, de forma que el compilador las pueda reconocer sin problema.

Lo vemos en el siguiente ejemplo:

```swift
var numero = 9 ; print("El número es \(numero)")
```

Ejemplo de entrada y salida en Swift

```swift
// Leo un número entero por teclado
print("Introduce un entero")
var enterostring = readLine()!
var numero = Int(enterostring)!
// Leo un numero real por teclado
print("\nIntroduce un número con decimales")
var realstring = readLine()!
var real = Double(realstring)!
// Mostramos los resultados
print("\nEl número entero es: \(numero)")
print("El número real es: \(real)")
```

Ejemplo de Entrada y Salida de Información en Swift

Introducción:

En este ejemplo práctico, abordaremos detalladamente la entrada y salida de información en Swift. El programa que desarrollaremos solicitará al usuario un **número entero**, un **número real** y una cadena, para posteriormente mostrarlos en la pantalla.

La instrucción 'print' será utilizada para imprimir mensajes en la pantalla, incluyendo la capacidad de introducir saltos de línea con '\n' y tabuladores con '\t'.

Código:

```
import Foundation
// Bloque 1: Lectura de un número entero
print("Introduce un entero")
var enterostring = readLine()!
var numero = Int(enterostring)!
// Bloque 2: Lectura de un número real
print("\nIntroduce un número con decimales")
var realstring = readLine()!
var real = Double(realstring)!
// Bloque 3: Lectura de una cadena
print("\nIntroduce una cadena")
var cadena = readLine()!
// Bloque 4: Mostrar resultados
print("\nEl numero entero es: \(numero)")
print("El numero real es: \(real)")
print("La cadena es: \(cadena)")
```

Explicación del Código:

- En el primer bloque, solicitamos y leemos un número entero, mostrando un mensaje al usuario y creando la variable 'enteroString'.

- En el segundo bloque, realizamos un proceso similar para un número real, utilizando casting a 'Double'.

- En el tercer bloque, leemos una cadena y la almacenamos en la variable correspondiente.

- En el cuarto bloque, mostramos los resultados utilizando la función 'print' y presentamos los valores de las variables.

Compilación y Ejecución:

Para compilar y ejecutar este ejercicio, utilizaremos el archivo 'shiftc.exe' ubicado en 'C:\shift\bin', proporcionándole la ruta del archivo como argumento.

Ejemplo de Ejecución:

Ejecutando el programa...

Introduce un número entero: 7

Introduce un número real: 5.3

Introduce una cadena: hola

Resultados:

Número entero: 7

Número real: 5.3

Cadena: hola

Este formato mejorado facilita la lectura y comprensión del código, proporcionando una estructura más clara para su estudio.

Arrays

En el lenguaje de programación Swift, los arrays son un tipo de dato básico.

El concepto de array sigue siendo exactamente el mismo que hemos estudiado en otras asignaturas, es decir, seguiremos pudiendo tener una colección de datos del mismo tipo y realizar operaciones sobre ellos.

Para declarar un array en Swift, tendremos que igualar una variable a una serie de valores colocados entre corchetes ([]) y **separados mediante comas**. También podremos declarar arrays que sean **constantes mediante let**.

Un ejemplo de declaración de un array puede ser el siguiente:

```
var array = [1, 2, 3, 4, 5]
```

Al igual que con las variables, **podremos indicar el tipo** de elementos que contendrá el array, aunque **no sea obligatorio**. Podremos hacerlo de la siguiente forma:

```
var palabras: [String] = ["Hola", "que", "tal"]

let palabrasconstantes: [String] = ["Buenos", "días"]
```

Se podrá **mostrar un array por pantalla directamente con la instrucción print** tratándolo como una **variable** normal y corriente.

Suma de arrays en swift:

let primera = ["A","B"]

let segunda = ["C","D"]

let tercera = primera + segunda

La operación + en Swift es utilizada para concatenar dos arrays, y en este caso, `tercera` contendrá los elementos de ambas arrays concatenados en orden.

Al ejecutar el código Swift:

var numeros = [1, 2, 3]

numeros += [4]

El array tendrá 4 elementos:

La operación '+=' se utiliza para agregar elementos a un array en Swift. En este caso, se agrega el número 4 al array 'números', por lo que después de la ejecución, el array contendrá los elementos [1, 2, 3, 4]. Sin embargo el test dice que el código dará error...

En Swift, el operador de incremento clásico ++ no está disponible. En su lugar, se recomienda utilizar la forma **+=1** para incrementar una variable en una unidad. El operador ++ ha sido eliminado en Swift, y se fomenta el uso de la sintaxis más explícita.

El lenguaje Swift nos ofrece una gran cantidad de métodos para poder operar con arrays, siendo algunos de ellos los que podemos ver en la siguiente tabla.

Métodos de los arrays en Swift

Función → Funcionalidad

count → Devuelve la cantidad de elementos del array.

append(valor) → Inserta un valor **al final** del array.

insert(valor, at: posición) → Inserta un valor en el array en la **posición indicada**.

remove(at: posición) → Elimina el elemento de la **posición indicada**.

Sort() → Ordena en orden ascendente el array.

sort(by: >) → Ordena en orden **descendente** el array.

Reversed() → **Invierte** los elementos del **array**.

Shuffle() → **Desordena** de forma aleatoria los elementos. Este método funciona a partir de Swift 4.2.

array[posición] = nuevovalor → Asigna un **nuevo valor** a la posición del array indicada.

print(array) → Imprime por consola el **array completo**.

Ejemplo de arrays

Introducción:

Presentamos a continuación un ejemplo práctico de cómo trabajar con Arrays en Swift. Comenzamos **declarando un Array** de enteros vacío.

Luego, se leen tres datos de tipo entero (número uno, número dos y número tres) que se agregan al Array utilizando el método 'append'.

El contenido del Array se muestra mediante 'print' antes y después de ordenarlo, con el objetivo de verificar la efectividad de la operación de ordenamiento.

Proceso:

1. Declaración del Array:

- Se inicia declarando un Array de enteros vacío utilizando la sintaxis

```
var Array: [Int] = []
```

2. Lectura de Datos:

- Se solicitan tres datos de tipo entero (número uno, número dos y número tres) al usuario mediante la función **readLine()**.

3. Agregar Datos al Array:

- Los datos leídos **se agregan** al Array usando el método **append**.

4. Visualización del Array:

- Se muestra el contenido del Array mediante '**print**' antes de realizar el ordenamiento.

5. Ordenamiento del Array:

- El Array se ordena utilizando el método '**sort**()'.

6. Visualización del Array Ordenado:

- Se muestra el **Array después de ordenarlo**, confirmando el éxito del proceso.

Compilación y Ejecución:

Se utiliza el compilador de swift para compilar y ejecutar el fichero del ejercicio. Durante la ejecución, se solicita al usuario ingresar tres números, y luego el programa muestra el Array antes y después de ordenarlo.

```
import Foundation

// Me declaro las variables
var numero = 0
var array: [Int] = []

// Introducimos tres elementos en el array
print("Introduce un entero")
var enterostring1 = readLine()!
var numero1 = Int(enterostring1)!

print("Introduce un entero")
var enterostring2 = readLine()!
var numero2 = Int(enterostring2)!

print("Introduce un entero")
var enterostring3 = readLine()!
var numero3 = Int(enterostring3)!

// Agregamos los elememtos al array
array.append(numero1)
array.append(numero2)
array.append(numero3)
```

```
print("El array es \(array)")

array.sort()

print("El array ordenado es \(array)")
```

resultado:

Introduce un entero

4

Introduce un entero

2

Introduce un entero

8

El array es [4, 2, 8]

El array ordenado es [2, 4, 8]