

Introducción a Swift

✓ **Diploma:** <https://platzi.com/p/rubseg/curso/1772-swift-5/diploma/detalle/>

¿Que es Swift? Es una forma fantástica de **escribir software**, ya sea **para teléfonos, computadoras de escritorio, servidores o cualquier otra cosa que ejecute código**. Es un lenguaje de programación seguro, rápido e interactivo que combina lo mejor del lenguaje moderno con la sabiduría de la cultura más amplia de ingeniería de Apple y las diversas contribuciones de su comunidad de código abierto. El compilador está optimizado para el rendimiento y el lenguaje está optimizado para el desarrollo, sin comprometer ninguno de los dos.

Un lenguaje para dominarlos a todos.

Variables y constantes

Las variables almacenan datos que pueden cambiar

```
var currentLoginAttempt = 0
```

Las constantes igual pero sin cambiar su valor

```
let maximumNumberOfLoggingAttempts = 3
```

Type annotations

Cuando una variable no se le da valor inicial hay que indicarle el tipo de dato que será posteriormente

```
5  
6 var welcomeMessage  
7  
8
```

✖ Type annotation missing in pattern

```
var welcomeMessage: String
```

```
welcomeMessage = "Bienvenido usuario"
```

```
var red, green, blue : Double
```

Nomenclatura de variables

Se recomienda camelCase

```
var myName = "rubén"
```

Imprimir valores con la función print

Se puede imprimir texto directamente

```
print("Hola")
```

Variables

```
print(myName)
```

Se puede interpolar texto y variables

```
print("El número de login actual es: \$(currentLoginAttempt)")
```

Comentarios

Son líneas de código que no se va a ejecutar, y sirven para dar más información sobre lo que hace el código.

```
// Comentario en línea
```

```
/*  
Comentario multilinea  
Podemos dar más información  
*/
```

Tipos de datos

Inferencia de tipos (type safe language):

Es un proceso por el cual el compilador determina el Tipo de una Variable, Constante, etc. que es definida sin una declaración explícita de su Tipo. El tipo es inferido a partir del valor asignado a esa Variable, Constante, etc.

Números

Enteros	Integer	let age: Int = 30
Flotantes (hasta 6 decimales)	Float	let timeRunF: Float = 6.5648339
Double (más de 6 decimales)	Double	let timeRunD: Double = 6.5648339

Conversión de tipos de dato

Podemos pasar variables de un tipo de dato a otro para poder operar entre ellas. Nosotros decidimos el tipo de conversión, en caso de número podemos llegar a truncar y perder datos.

```
let twoThousand: UInt16 = 2_000
let one: UInt8 = 1
let twoThousandOne = twoThousand + UInt16(one)
```

```
print("Suma \(twoThousand) y \(one) = \(twoThousandOne) con conversión de tipo de dato")
```

```
let pi: Double = 3.14159
let integerPi = Int(pi)
```

```
print("Pi \(pi) se convierte a int y es truncado: \(integerPi)")
```

Booleanos

El Tipo de Dato Booleano sólo puede tomar dos valores: true o false (verdadero o falso). Para definir este Tipo se utiliza la palabra reservada Bool.

Sería igual que en otros lenguajes.

```
var userLogged: Bool = false

if userLogged {
    print("User logged: \(userLogged)")
} else {
    print("User logged: \(userLogged)")
}
```

Cadenas y caracteres

En el siguiente curso se amplía todo !

Tuplas

Aglutinan varios valores en una sola variable, como si fuese clave-valor.

```
let http404Error = (404, "Página no encontrada")
let (statusCode, statusMessage) = http404Error

print("El código del estado es \(statusCode)")
print("El mensaje del servidor es \(statusMessage)")
print("El código de error es \(http404Error.0) y el mensaje es \(http404Error.1)")

let http200Status = (statusCode: 200, statusMessage: "OK")
print("El código de estado es \(http200Status.statusCode) y el mensaje es \(http200Status.statusMessage)")
```

Type Alias

Permite renombrar los tipos de datos.


```
typealias AudioSample = UInt16  
let bookAudio: AudioSample = AudioSample.max
```

Manejo de optional

Optional y Nil

En Swift no existen valores nulos, si ausentes de valor.

```
8  var possibleInt = "Sauron"  
9  var convertInt = Int(possibleInt) // int? Opcional  
10  
11 var possibleInt = "31"  
12 var convertInt = Int(possibleInt) // int? Opcional  
13
```

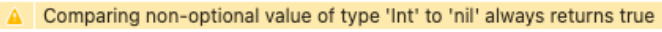


```
var serverResponse: Int? = 200 // si no estamos seguro del valor podemos ponerlo  
opcional  
serverResponse = nil
```

Force unwrapping de una variable optional

Si tenemos la seguridad de que una Variable Opcional tiene siempre valor entonces podremos acceder al mismo haciendo un Forced Unwrapping añadiendo una exclamación ! al nombre de la variable Se puede obligar con ! pero eso haría que se rompa la app ya que nunca puede ser null un valor

```
7  if convertedAge != nil {  
8      print("La edad no es nula \(convertedAge)")  
9  } else {  
10     print("La edad es nula")  
11 }  
12
```



Optional binding

Esta técnica nos permite asegurarnos que una variable Opcional tiene valor, y si lo tiene podremos extraerlo para su uso posterior.

```
var nextWin: String?  
nextWin = "33"
```

```
if let victory = nextWin {  
    // como se ha podido inicializar victory con el valor de nextWin  
    // sabemos que nextWin no es NIL
```

```
}
```

Unwrap implícito

Podremos hacer un desencapsulamiento implícito a la hora de declarar un Opcional siempre que tengamos la seguridad de que la Variable o Constante va a tener un valor.

Los Opcionales así declarados no tendrán que ser desencapsulados para poder acceder al valor que almacenan.

Si no asignamos ningún valor a la Variable o Constante Opcional que definimos desencapsulada implícitamente, nuestra aplicación tendrá un provocar un error en tiempo de ejecución.

```
let possibleString: String? = "string opcional"  
let forcedString: String = possibleString!
```

Gestión de errores

Uso de try-catch

Para controlar y gestionar errores que pueden provocarse en el código.

```
func canThrowError() throws {  
    // función que puede lanzar un error  
}  
  
do {  
    try canThrowError() // intenta llamar a la funcion  
    // si llega aquí no ha habido error  
} catch {  
    // si llega aquí si hay error  
}
```

Aserciones

Se suelen utilizar para hacer mas robusto el código y chequear los valores.

Es algo muy útil en fase de desarrollo.

```
16 let age = -5  
17  
18 assert(age >= 0, "La edad de una persona no puede ser menor que cero") // esto sale en consola error: Exec... Error  
19 // ...el código sigue...  
  
_lldb_expr_99/05_gestion_de_errores.playground:18: Assertion failed: La edad de una persona no puede ser menor que cero
```

