

Programación en Swift

✓ **Diploma:** <https://platzi.com/p/rubseg/curso/1774-swift/diploma/detalle/>

Condicionales y operaciones básicas

Operaciones de asignación y aritméticas

Para realizar una asignación con =

Para crear o cambiar el valor de una variable

```
let a = 10;
```

```
var b = a;
```

Para comparar variables es con ==

```
if a == b {  
    print("los valores \ (a) y \ (b) son iguales")  
}
```

Operadores aritméticos

5+3

13-9

8*2

10/2

Para obtener el resto %

17 9 % 4	1
----------	---

Incrementar +=

Decrementar -=

```
a += 5
```

```
b -= 1
```

Comparaciones

22	<code>1 == 1</code>	<input type="checkbox"/> true
23	<code>1 == 2</code>	<input type="checkbox"/> false
24		
25	<code>1 != 2</code>	<input type="checkbox"/> true
26		
27	<code>2 > 1</code>	<input type="checkbox"/> true
28	<code>2 < 1</code>	<input type="checkbox"/> false
29	<code>2 >= 1</code>	<input type="checkbox"/> true
30	<code>1 >= 1</code>	<input type="checkbox"/> true
31	<code>2 >= 1</code>	<input type="checkbox"/> true

Operaciones ternarias

Se realizan con el ?

```
var logged = false
var message = logged ? "wellcome" : "You need login"
print(message)
```

Operador Nil

Para no tener que hacer esto

```
var otherCarKm = (renaultKm != nil ? renaultKm : defaultKm)
```

En swift podemos usar ?? para controlar ese nil en todo momento y mejorar la legibilidad del código

```
let defaultKm = 0
var toyotaKm = 123987
var renaultKm: Int?
```

```
var myCarKm = renaultKm ?? defaultKm
```

Rangos

Son atajos para tener un rango de valores

Tenemos los rangos cerrados que incluyen valor de inicio y fin

```
for i in 1...5 {
    print(i)
}
```

Semiabiertos incluye el valor de inicio pero no la final

```
for i in 1 ..< 5 {  
    print(i)  
}
```

Y abiertos

```
var names = ["Rubén", "María", "Idril"]
```

```
for name in names[1...] {  
    print(name)  
}
```

```
for name in names[..<2] {  
    print(name)  
}
```

Operadores lógicos

AND - &&

Y Lógico. Es verdadero (true) cuando los dos operandos son verdaderos, y falso (false) en cualquier otro caso

```
let enterDoorCode = true  
let passRetinaScan = false  
  
if enterDoorCode && passRetinaScan {  
    print("Se cumplen las 2 condiciones.. puedes entrar")  
} else {  
    print("Acesso denegado")  
}
```

OR - ||

O Lógico. Es verdadero (true) cuando cualquiera de los dos operandos son verdaderos, y falso (false) cuando los dos operandos son falsos

```
if enterDoorCode || passRetinaScan {  
    print("Se cumple alguna de las identificaciones.. puedes entrar")  
} else {  
    print("Acesso denegado")  
}
```

NOT - !

No Lógico. Invierte un valor booleano, es decir, true se convierte en false y false se convierte en true

```
let allowEntry = false
if !allowEntry {
    print("Acceso denegado..")
}
```

Manipulación de cadenas

String

Una cadena es una serie de caracteres, como "hola, mundo". Las cadenas Swift están representadas por el tipo String. Se puede acceder al contenido de una cadena de varias maneras, incluso como una colección de valores de caracteres.

Inicialización

Existen varias formas de inicializar los tipo String.

```
var someString = "Soy un string cualquiera"
```

```
var multiline = """
    Así es como se hace\
    una cadena de varias líneas
    """
```

```
var emptyString = ""
```

```
var anotherEmptyString = String()
```

Podemos escribir caracteres especiales dentro del string con el back slash \ y el Unicode [https://en.wikipedia.org/wiki/Basic_Latin_\(Unicode_block\)](https://en.wikipedia.org/wiki/Basic_Latin_(Unicode_block))

```
var wiseWord = "\"La imaginación es mas importantes que el
saber\" - Albert Einstein"
```

```
let dolarSign = "\u{24}"
```

Podemos añadir mas texto a un string

```
var newSomeString = "La 33"
newSomeString += " en el año 2024"
```

char

Solo soportan un carácter y se pueden utilizar para operar de diferentes formas solos o junto a strings.

```

let dogName = "Idril 🐕 🧑"
for character in dogName {
    print(character)
}

print("\(dogName.count) caracteres")

let exclamationMark = "!"

let nameChars: [Character] = ["I","d","r","i","l"]

var nameString = String(nameChars)

```

Substrings

Parar sacar una parte de un String

```

let index = greeting.firstIndex(of: ",") ??
greeting.endIndex // ?? puede ser nil
print(index)

```

```

let firstPart = greeting[..<index] // esto no sería un
string

```

```

let newString = String(firstPart) // esto si sería un string

```

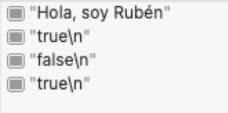
Prefijos y sufijos

Podemos por ejemplo buscar si tiene un prefijo o sufijo un string.

```

let newGreeting = "Hola, soy Rubén"
print(newGreeting.hasPrefix("Hola"))
print(newGreeting.hasSuffix("Adiós"))
print(newGreeting.hasSuffix("n"))

```



The output of the code is as follows:

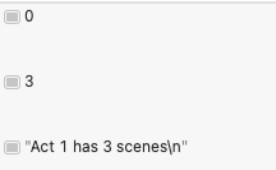
- "Hola, soy Rubén"
- "true\n"
- "false\n"
- "true\n"

En el siguiente ejemplo buscamos el número de escenas en el acto 1

```

79 var act1SceneCount = 0
80 for scene in collection {
81     if scene.hasPrefix("Act 1") {
82         act1SceneCount += 1
83     }
84 }
85 print("Act 1 has \(act1SceneCount) scenes")

```



The output of the code is as follows:

- 0
- 3
- "Act 1 has 3 scenes\n"

Estructuras de datos

Swift proporciona tres tipos de colecciones principales, conocidos como **matrices**, **conjuntos** y **diccionarios**, para almacenar colecciones de valores.

Las matrices son colecciones ordenadas de valores.

Los conjuntos son colecciones desordenadas de valores únicos.

Los diccionarios son colecciones desordenadas de asociaciones clave-valor.

Arrays

Una matriz almacena valores del mismo tipo en una lista ordenada. El mismo valor puede aparecer en una matriz varias veces en diferentes posiciones.

Hay diferentes formas de crear arrays.

```
var someInts = [Int]()
someInts.append(23)
someInts.append(28)
```

```
someInts = []
```

```
someInts = [55,3,96,675,12]
```

```
var someDouble = Array(repeating: 3.14, count: 5)
someDouble.count
```

```
var moreDoubles = Array(repeating: 2.5, count: 3)
```

```
var aLotOfDoubles = someDouble + moreDoubles
aLotOfDoubles.count
```

```
var shoppingList : [String] = ["Manzanas", "Cebollas",
    "Pechuga de pollo", "Queso", "Atún"]
```

Acceso y manipulación de arrays

Añadir al final un elemento

```
shoppingList.append("Cerveza")
```

```
shoppingList += ["Guacamole", "Totopos"]
```

Añadir indicando la posición

```
shoppingList[0] = "Huevos"
```

Acceder a un elemento

```
var firstElement = shoppingList[0]
```

Acceder a un rango de elementos

```
shoppingList[4...6]
```

Eliminar por posición

```
shoppingList.remove(at: 1)
```

Eliminar el último elemento o el primero

```
shoppingList.removeLast()
```

```
shoppingList.removeFirst()
```

Iterando arrays

```
for item in shoppingList {  
    print(item)  
}
```

Otra forma de iterarlo y poder mostrar el número que ocupa

```
for (idx, item) in shoppingList.enumerated() {  
    print(idx, item)  
}
```

Conjuntos

Set almacena valores distintos del mismo tipo en una colección sin un orden definido.

Puede utilizar un conjunto en lugar de una matriz cuando el orden de los elementos no es importante o cuando necesita asegurarse de que un elemento solo aparezca una vez.

Iteraciones y operaciones sobre conjuntos

```
var letters = Set<Character>()  
letters.insert("a")  
letters.insert("b")  
letters.insert("c")  
print(letters)
```

```
var favouriteGames : Set<String>
```

```
favouriteGames = ["Assassins Creed", "The Witcher 3", "The  
last of us"]  
print(favouriteGames)
```

Añadir un elemento

```
favouriteGames.insert("God of War")  
print(favouriteGames)
```

Eliminar un elemento

```
favouriteGames.remove("The last of us")  
print(favouriteGames)
```

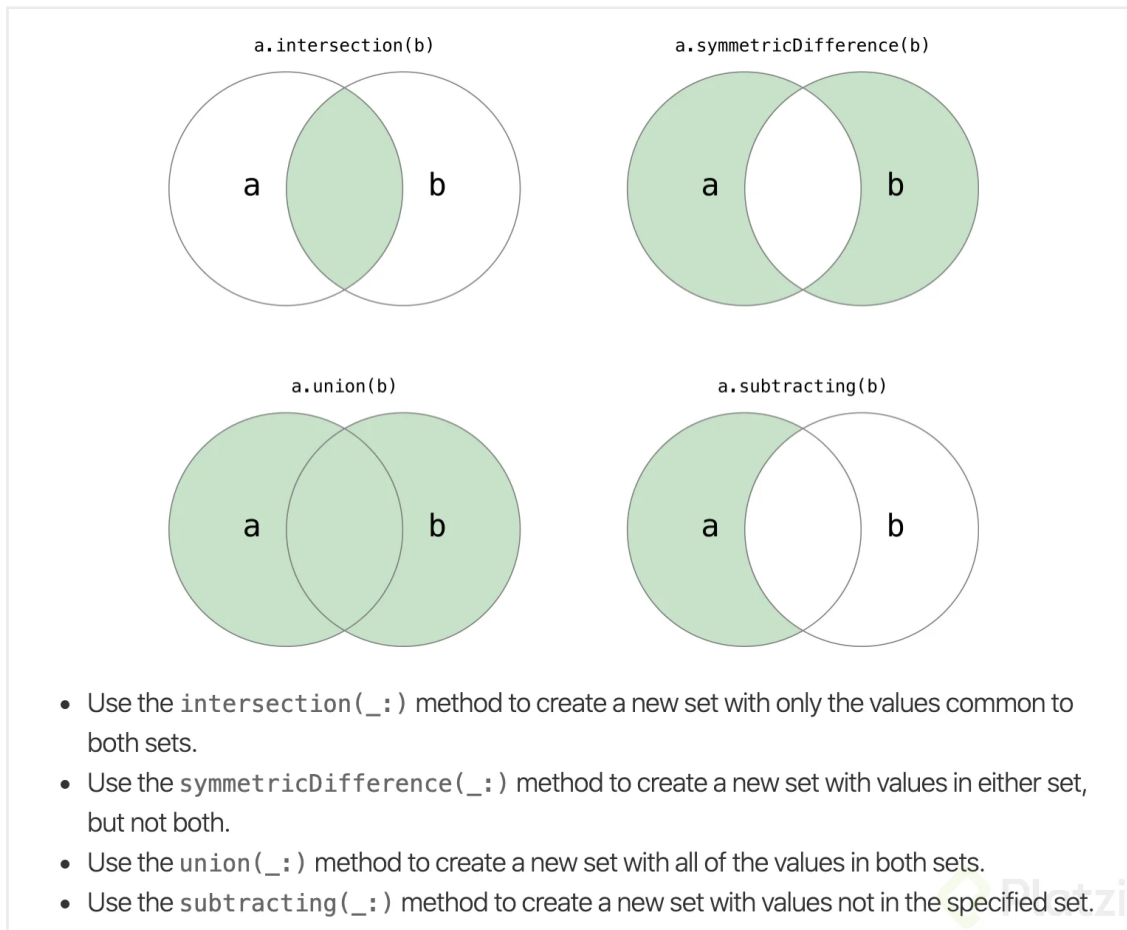
Comprobar si contiene un elemento

```
if favouriteGames .contains("Assassins Creed") {  
    print("Me encanta ese juego")  
}
```

Recorrer el set

```
for vg in favouriteGames.sorted() {  
    print(vg)  
}
```

Tipos de operaciones con conjuntos



Diccionarios

Un diccionario almacena asociaciones entre claves del mismo tipo y valores del mismo tipo en una colección sin un orden definido.

Cada valor está asociado con una clave única, que actúa como identificador de ese valor dentro del diccionario.

A diferencia de los elementos de una matriz, los elementos de un diccionario no tienen un orden específico.

Se utiliza un diccionario cuando necesita buscar valores según su identificador, de la misma manera que se utiliza un diccionario del mundo real para buscar la definición de una palabra en particular.

```
var isWorking = [String : Bool]()
isWorking["Maria"] = true
isWorking["Ruben"] = false
isWorking["Idril"] = false
```

```
var airports: [String : String] = ["YYZ" : "Toronto",
                                   "DUB" : "Dublin",
```

```
        "PMI" : "Palma de  
Mallorca"]  
  
if let airportName = airports["PMI"] {  
    print("El aeropuerto PMI es \(airportName)")  
}
```

Iteración de diccionarios

Iterar clave-valor

```
for (key, value) in airports {  
    print("\(key) - \(value)")  
}
```

Iterar claves

```
for airportKey in airports.keys {  
    print("Airport key: \(airportKey)")  
}
```

Iterar valores

```
for airportName in airports.values {  
    print("Airport name: \(airportName)")  
}
```

Obtener claves o valores en una lista

```
let airportKeys = [String](airports.keys)  
let airportNames = [String](airports.values)
```

Sentencias de control

For-in

El bucle for-in se utiliza para iterar sobre una secuencia, como elementos de una matriz, rangos de números o caracteres de una cadena.

Algunos ejemplos.

```
let names = ["Rubén", "María", "Idril"]  
  
for name in names {  
    print("Hola \(name)")  
}
```

```
let numberOfLegs = ["spider": 8, "ant": 6, "dog": 4]  
  
for (animalName, legCount) in numberOfLegs {
```

```
        print("Animal \(animalName), number of legs \
(numberOfLegs)")
    }

    for idx in 1...10 {
        print(idx)
    }

    let base = 2
    let power = 10
    var answer = 1

    for _ in 1...power {
        answer *= base
        print("power of 2 to \(power) : \(answer)")
    }
```

While

Un bucle while realiza un conjunto de declaraciones hasta que una condición se vuelve falsa.

Este tipo de bucles se utilizan mejor cuando no se conoce el número de iteraciones antes de que comience la primera.

Swift proporciona dos tipos de bucles while:

- while evalúa su condición al comienzo de cada paso por el bucle.
- repeat-while evalúa su condición al final de cada paso por el bucle.

Algunos ejemplos.

```
i = 0
repeat {
    i+=1
} while i <= 10
print(i)
```

If

Swift proporciona dos formas de agregar ramas condicionales a su código: la declaración if y la declaración switch. Normalmente, se utiliza la declaración if para evaluar condiciones simples con solo unos pocos resultados posibles.

En su forma más simple, la declaración if tiene una única condición if. Ejecuta un conjunto de declaraciones sólo si esa condición es verdadera.

La declaración `if` puede proporcionar un conjunto alternativo de declaraciones, conocida como cláusula `else`, para situaciones en las que la condición `if` es falsa. Estas declaraciones están indicadas por la palabra clave `else` y son opcionales.

Algunos ejemplos.

```
var temp = 37
if temp <= 15 {
    print("Hace frío: se enciende la calefacción")
} else if temp >= 25 {
    print("Hace mucha calor: se enciende el aire
condicionado")
} else {
    print("La sensación térmica es agradable, no hace falta
modificarla")
}
```

Switch

La estructura `switch` considera un valor y lo compara con varios patrones coincidentes posibles. Luego ejecuta un bloque de código apropiado, basado en el primer patrón que coincide exitosamente.

En su forma más simple, `switch` compara un valor con uno o más valores del mismo tipo.

```
let someCharacter: Character = "H"
switch someCharacter.lowercased() {
    case "a":
        print("Es la primera letra del alfabeto")
    case "z":
        print("Es la última letra del alfabeto")
    default:
        print("Es alguna otra letra")
}
```

Con intervalos.

```
let lunasDeSaturno = 62
let frase = "Lunas de Saturno"
var naturalCount: String
switch lunasDeSaturno{
    case 0:
        print("No hay lunas")
    case 1..<5:
        print("Hay pocas lunas")
}
```

```

    case 5..<100:
        print("Muchísimas lunas")
    default:
        print("Error!")
}

```

Con tuplas.

```

let somePoint = (0,1)
switch somePoint{
case (0,0):
    print("El punto \(somePoint) es el origen de
coordenadas")
case (_,0):
    print("El punto \(somePoint) se halla sobre el eje X de
las coordenadas")
case (0,_):
    print("El punto \(somePoint) se halla sobre el eje Y de
las coordenadas")
default:
    print("El punto \(somePoint) se halla en algún otro
lado")
}

```

Sentencias de transferencia de control

Continue y break

La instrucción continue le dice al bucle que detenga lo que está haciendo y comience de nuevo al comienzo de la siguiente iteración del bucle. Dice "He terminado con la iteración del bucle actual" sin salir del bucle por completo.

```


let sentence = "Goku siempre gana a sus adversarios"
let charactersToRemove:[Character] = ["a", "e",
    "i", "o", "u"]
var filteredSentence = ""
for ch in sentence {
    if charactersToRemove.contains(ch) {
        continue
    }
    filteredSentence.append(ch)
}
print(filteredSentence)

```

```
Gk smpr gn ss dvrsrs
```

La declaración de `break` finaliza inmediatamente la ejecución de una declaración de flujo de control completa.

```
let sentence = "Goku siempre gana a sus adversarios"
let charactersToRemove:[Character] = ["a", "e",
    "i" , "o" , "u"]
var filteredSentence = ""
for ch in sentence {
    if charactersToRemove.contains(ch) {
        continue
    }
    filteredSentence.append(ch)
    if ch == "d" {
        break
    }
}
print(filteredSentence)
```



Gk smpr gn ss d

Return y guard

```
var people : [String : Any] = ["name" : "Alberto Moreno",
    "age" : 23, "isMale" : true]
var moraPeople = ["name" : "Alberto Moreno", "age" : 23,
    "isMale" : true] as [String : Any]

func testUserValidation(person:[String: Any]) {
    guard let surname = people["name"] else {
        // no existe surname
        return
    }
    print(surname)
}
// si se ha creado surname
```

Available API: Manejo de versiones

Para comprobar si existe algo o no en diferentes SO de Apple y ramificar el código para que por plataforma haga unas sentencias u otras.

```
if #available(iOS 12, *) {
    // Ejecutar acciones posteriores a iOS 12 o posterior
```

}