Desarrollo de aplicaciones para dispositivos móviles iOS

Tema 1 - Aspectos básicos de los playgrounds

INSTRUCTOR: Eduardo García Solis
lalogarciasolis@gmail.com

Marco contextual - Swift

- Creado por Apple y lanzado en 2014
- Versión actual 5.1
- Para iOS, macOS, tvOS, iPadOs, watchOS y server-side

En palabras de Apple, Swift es[1]

```
class 🐶🐶
{
        func 🤷(😎: Int, 😻: Int) -> Int
        {
                return 😎 + 😻
        }
}

var 🐔 = 3
var 😥 = 🐔 + 2

var 🐶 = 🐶🐶()
```

Versátil lenguaje de programación para teléfonos, aplicaciones de escritorio y servidores o cualquier lugar donde se ejecute código. Es seguro, rápido y combina lo mejor de un lenguaje de programación moderno con la cultura de ingeniería de Apple y las contribuciones de su comunidad. El compilador está optimizado para la ejecución y el lenguaje para el desarrollo sin comprometer ninguno de los aspectos mencionados.

Es amigable para los nuevos programadores. Escribir código de Swift en un playground permite experimentar con código y ver los resultado inmediatamente sin volver a compilar o ejecutar en un aplicación.

Define tipos de errores comunes (para ayudar a evitarlos) en programación mediante la adopción de patrones de programación de errores:

- Variables siempre son inicializadas antes de usar
- Los índices de los arreglos son verificados para evitar salirse del límite
- Los números enteros son verificados para no desbordarse

- Los Opcionales aseguran que el valores nulos sean manejados explícitamente
- La memoria es administrada automáticamente
- El manejo de errores permite controlarlos y recuperarse de fallos inesperados

¿Qué es un playground?

Concepto del idioma inglés que se puede traducir como “patio de recreo”. Es un ambiente listo para programar y realizar pruebas sobre Swift. Se puede escribir, experimentar con nuevas sintaxis y probar el desempeño de un algoritmo desde un playground, todo sobre el IDE Xcode.

Antes de continuar

El lenguaje Swift tiene más funcionalidades de las que presenta este documento, algunas de ellas son:

- Opcionales
- Otros usos de Switch
- Otros usos de For
- Otros usos de Rangos
- Manejo de errores

Algunas de ellas se verán más adelante pero otras no. Para profundizar en el uso del lenguaje Swift se puede consultar la guía de referencia: Swift Book, parte de los siguientes temas están desarrollados con dicha guía.

Los siguientes ejemplos se pueden ejecutar en un playground de Xcode, para eso es necesario una Mac y dicho programa. También, puedes practicar en páginas web donde puedes ejecutar código de Swift, aquí una de ellas Online Swift Playground.

1.1 Aspectos básicos de los playgrounds[2]

Identificadores

Nombre único dentro del paquete, clase, función o bloque de código que se le asignará a una variable, constante, parámetro, tipo de dato, clase o función. La declaración en Swift de hace con la siguiente regla:

```
[var, let, func, etc] [NOMBRE_IDENTIFICADOR]:[TIPO_DE_DATO_ES_OPCIONAL] =
[VALOR_QUE_SE_ALMACENA]
```

Ejemplos:

```
var miVariable = 42
let miConstante = 42
let miConstante:Float = 42
func miFuncion () {}
```

Variables

Espacio de memoria que almacenará un valor, este valor puede cambiar a través del tiempo de vida de la aplicación, se hace referencia a él mediante un identificador. Swift puede inferir el tipo de dato o el programador lo puede hacer de forma explícita. Se usa la palabra reservada "var" para su declaración.

```
//Se infiere que el tipo de dato es entero
var miVariable = 42

//Se declara de forma explícita que es un tipo de dato MiTipoDeDato
var miFuncion:MiTipoDeDato = 70

//Se infiere que el tipo de dato es entero al tener el mismo valor almacenado en
//miVariable
var miFuncion = miVariable
```

Constantes

Espacio de memoria que almacenará un valor, este valor NO puede cambiar de tal motivo que se mantendrá constante durante el tiempo de vida de la aplicación, se hace referencia a él mediante un identificador. Swift puede inferir el tipo de dato o el programador lo puede hacer de forma explícita. Se usa la palabra reservada "let" para su declaración.

```
//Se infiere que el tipo de dato es entero
let miConst = 42

//Se declara de forma explícita que es un tipo de dato MiTipoDeDato
```

```
let miConstConTipoDeDato:MiTipoDeDato = 70

//ESTO ES UN ERROR NO SE PUEDE CAMBIAR EL VALOR
miConst = miConstConTipoDeDato
```

Tipos de datos[3]

Es la propiedad de un valor que determina su dominio (qué valores puede tomar), qué operaciones se le pueden aplicar y cómo es representado internamente por el programa. Todo valor (variable o constante) tiene tipo de dato, este se puede indicar de forma explícita o Swift lo puede inferir de acuerdo al valor que se almacenará.

```
//Se infiere que el tipo de dato es entero, pues 42 es un entero
var miVariable = 42

//Se infiere que el tipo de dato es double, pues termina con .0
var miVariableDouble = 42.0

//Se especifica que el tipo de dato es Double a pesar de que 42 es un entero
var miVariableExplicitDouble:Double = 42

//Tipo de dato MiTipoDeDato
var miFuncion:MiTipoDeDato = MiTipoDeDato()
```

Para imprimir o concatenar el valor (incluye String) de una variable o constante en una cadena se hace uso de la interpolación. Para esto se pone el identificador dentro de paréntesis y anteponiendo una diagonal inversa [4]:

```
let apples = 3
let oranges = 5
let appleSummary = "I have \(apples) apples."
let fruitSummary = "I have \(apples + oranges) pieces of fruit."
```

1.2 Operadores

Los operadores son símbolos que indican cómo se deben manipular los operandos. Los operadores junto con los operandos forman una expresión, que es una fórmula que define el cálculo de un valor. Los operandos pueden ser constantes, variables o llamadas a funciones, siempre que éstas devuelven algún valor. El compilador evalúa los operadores, algunos de izquierda a derecha, otros de derecha a izquierda, siguiendo un orden de precedencia. Este orden se puede alterar utilizando paréntesis para forzar al compilador a evaluar primero las partes que se desean [5].

Como otros lenguajes Swift maneja los típicos operadores clasificados de la siguiente manera [6]:

Asignación

```
let mensaje = "hola"
```

Aritméticos

```
let area = Double.pi * (radius * radius) - (4 + 5) / (10 & 35)
```

Operadores Aritméticos	Descripción
+	Suma dos números.
-	Resta dos números.
*	Multiplica dos números.
/	Divide dos números.
&	El resto de una división.

Relacionales o de comparación

```
let isVerdadero: Bool = 4 < 8
```

Operadores Relacionales	Descripción
<	Evalúa si el número de la izquierda es más pequeño que el número a la derecha.
<=	Evalúa si el número de la izquierda es más pequeño o igual al número a la derecha.
>	Evalúa si el número de la izquierda es mayor al número a la derecha.
>=	Evalúa si el número de la izquierda es mayor o igual al número a la derecha.
==	Evalúa si el número de la izquierda es igual al número a la derecha.
!=	Evalúa si el número de la izquierda no es igual al número a la derecha.
===	Evalúa si ambas instancias apuntan a la misma referencia.
!==	Evalúa si ambas instancias no apuntan a la misma referencia.

Logicos

```
if (isBoletoGanador && isBoletoRifaAvion)
      print("Tienes el avión que no tiene ni Obama")
```

Operadores Lógicos	Descripción
&&	AND Lógico: verdadero solo si ambos son verdaderos, falso en caso contrario.
||	OR Lógico: verdadero si alguno es verdadero, falso solamente si ambos son falsos.
!	NOT Lógico: si es verdadero se torna en falso y viseversa.

Ternario

```
let isGanador = isBoletoGanador && isBoletoRifaAvion ?
      true
      :
      false
```

De rango

```
for index in 1..5 {
      print("Este mensaje será impreso 5 veces")
}
```

1.3 Funciones, parámetros y resultados

Una función es un bloque de código que puede o NO recibir valores (parámetros) y retornar o NO un valor. Su virtud reside en que se puede mandar a llamar las veces que se requiera para ejecutar las instrucciones que contiene y con los parámetros deseados. Al igual que las constante y variables tiene un identificador único dentro de su contexto (clase, playground, etc.). Los parámetros comúnmente determinan el valor que se obtendrá en el retorno. La estructura básica de las funciones es:

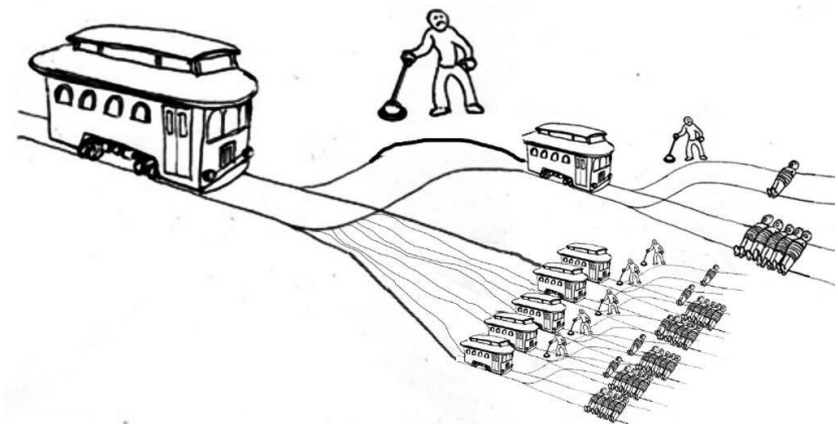
```
func [NOMBRE_FUNCION]([NOMBRE_PARAM_OPCIONAL: TipoDato]) -> [TIPO_RETORNO] {
   return [ALGUN_VALOR_TIPO_RETORNO]
}

Ejemplo 1:
//Función de nombre greet que recibe 2 parámetros. El primero con nombre person
//de tipo String y otro de nombre day igual de tipo String. El tipo de retorno
//es String
func greet(person: String, day: String) -> String {
   return "Hello \(person), today is \(day)."
}
greet(person: "Bob", day: "Tuesday")

Ejemplo 2:
//Función que recibe 2 números enteros y retorna el resultado de la suma de
//Ambos. El primer parámetro dentro de la función se llama num1 y el segundo
//num2. El guión bajo es para que en la llamada de la función se omita el
//nombre del parámetro
func suma(_ num1: Int, _ num2: Int) -> Int {
   return num1 + num2
}
suma(10, 20)

//Función sin parámetros y retornos
func funcionSinRetorno () {
      print ("no hay resultado")
}
```

1.4 Estructuras de control: if, switch, for, while[7]



Bloques de código que de acuerdo a condicionales determinan la sección del código a ejecutar o el número de veces. Las estructuras se pueden embeber una dentro otra, sin importar cantidad, orden y/o tipo. Se componen de la condición y el bloque a ejecutar si la condición es válida.

If

Es la forma más simple de las estructuras, como su nombre lo indica es una simple condición “Sí”. Sí la condición se cumple el bloque interno será ejecutado.

```
if (2 < 10) {
      //La condición se ha cumplido
      print (“2 es más pequeño que 10, la condición se ha cumplido”)
}
```

If-else

Funciona de la misma forma que IF, solo que sí la condición no se cumple se ejecutará otro código (else).

```
if (12 < 10) {
      //¡AQUÍ NO!
      print (“esto NO se va imprimir”)
} else {
      //¡AQUÍ SI!
      print (“12 NO es más pequeño que 10, la condición NO se cumplió, así que
            se imprime esto”)
}
```

Switch

Una sentencia Switch considera un valor y lo compara contra posibles coincidencias, el bloque que se ejecutará será el primero que empate con el valor comparado. Es una alternativa a IF con múltiples valores potenciales.

```
let someCharacter: Character = "z"
switch someCharacter {
      case "a":
         print("someCharacter coincide con ‘a’")
      case "z", "y":
         print("someCharacter coincide con ‘z’ o ‘y’")
      default:
         print("someCharacter NO coincide con ningún valor")
}
```

While

Esta estructura permite la ejecución de un bloque de sentencias mientras que la condición sea verdadera. Se utiliza cuando el número de iteraciones no es conocido ni siquiera si será ejecutada al menos una vez.

```
var eCont: Int = 0
```

```
//Mientras eCont sea menor que 10, el bloque se va ejecutar
while eCont < 10 {
      print (“Esta sentencia se ejecutará 5 veces”)
      eCont = eCont + 2
}
```

Repeat-While

Funciona igual que While, la única diferencia es que el bloque interno de sentencias se ejecutará al menos una vez. En otros lenguajes es conocido como do-while. El bloque se ejecutará mientras la condición sea verdadera.

```
var eCont: Int = 0
repeat {
      print ("Esta sentencia se ejecutará 5 veces 1")
      eCont = eCont + 2

//Mientras eCont sea menor que 0, se va ejecutar. Pero siempre al menos una vez
} while eCont < 0
```

For-In

En Swift se usa el ciclo for-in para iterar sobre una secuencia como arreglos, diccionarios, rango de números o caracteres de una cadena. Desde la versión 3 de Swift, no funciona como el tipo for de C (valor, condición e incremento).

Ejemplo 1:

```
//Arreglo de cadenas
let names = ["Anna", "Alex", "Brian", "Jack"]
//Se itera sobre el arreglo “names” dentro del ciclo cada elemento será llamado como
//”nombre”
for nombre in names {
      //El mensaje se ejecutará 4 veces, 1 por cada nombre
      print("Hello, \(nombre)!")
}
```

Ejemplo 2:

```
//Se itera sobre un rango del 1 al 5, empezando desde el 1
for index in 1...5 {
      print("Esta es la iteración número \(index)")
}
```

Guardas

Las guardas son como las estructuras if que ejecutan sentencias de acuerdo al resultado de una expresión booleana. Comúnmente se usan las guardas cuando se requiere que una condición se cumpla para continuar con la ejecución (validación de datos). A diferencia del if las guardas siempre tienen su sección else, este código será ejecutado si la condición NO se cumple.

```
func bienvenida(persona: Usuario) {
      //Se valida que el usuario esté autenticado
      guard let isAuth = persona.isAuth else {

            //Si la condición anterior no se cumple, ejecutamos este código
            print("Por favor inicie sesión")

            //Terminamos la ejecución de la función
            return
      }
      print("Hola \(persona.nombre)!")
}
```

1.5 Estructuras[8]

Las estructuras en Swift son bloques de código funcionales, pueden almacenar propiedades (variables y/o constantes) y métodos (funciones). En Swift, las estructuras son muy parecidas a las clases, las estructuras se distinguen de las clases al NO contar con características tales como herencia, poder hacer casting para identificar el tipo de la instancia, entre otras.

```
//Declaramos la estructura
struct EstructuraPruea {
      //Atributos de la estructura
      var estoEsUnaPropiedad = "Soy una propiedad de tipo cadena"
      var aquiOtraPropiedad:Int = 0

      //Métodos de la estructura
      func mostrarMensaje() {
        print("Este es el método de una Estructura")
      }

      func imprimeUnaPropiedad () {
        print("Este es el valor de mi propiedad: \(estoEsUnaPropiedad)")
      }
}
//Creamos estructura del tipo EstructuraPrueba
```

```
var res = EstructuraPruea()

//Llamamo sus métodos
res.mostrarMensaje()
res.imprimeUnaPropiedad()
```

1.6 Enumeraciones[9]

Una enumeración define un tipo común de valores relacionados y posibilita trabajar con esos valores de una forma segura (se evita errores) y funcional (se puede integrar en variables y/o estructuras de control) dentro de tu código.

```
//Así se declara un enumeración
enum PuntoCardinal {
   case norte
   case sur
   case este
   case oeste
}
//Variable que guarda el valor de un punto cardenal
var puntoCardinal = PuntoCardinal.norte

//La variable al ser del PuntoCardinal, se puede abreviar su asignación con .sur
puntoCardinal = .sur

//Se puede usar en un Switch con sus propios valores
switch puntoCardinal {
      case .norte:
         print("Fierro pariente")
      case .sur:
         print("Siempre que voy al sur, siento que bajo")
      case .este:
         print("Este, no éste ni esté")
      case .oeste:
         print("Spaghetti Western")
}
```

Referencias

1. About Swift — The Swift Programming Language
2. https://learnappmaking.com/xcode-playground-get-started-with-swift/
3. http://progra.usm.cl/apunte/materia/tipos.html
4. https://medium.com/coding-blocks/data-types-in-swift-where-it-all-starts-c1311fa93368
5. http://decsai.ugr.es/~jfv/ed1/c/cdrom/cap2/cap26.htm
6. https://swiftbycoding.dev/swift/operadores/

7. https://docs.swift.org/swift-book/LanguageGuide/ControlFlow.html
8. https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html
9. https://docs.swift.org/swift-book/LanguageGuide/Enumerations.html