Manual Básico para Aprender Swift

Swift es un lenguaje de programación desarrollado por Apple, diseñado para crear aplicaciones para iOS, macOS, watchOS y tvOS. Es un lenguaje poderoso, fácil de aprender y con un enfoque en la seguridad y el rendimiento.

Este manual cubrirá los conceptos básicos para comenzar a trabajar con Swift, desde la instalación hasta las estructuras de datos más comunes como arreglos (arrays) y diccionarios (dictionaries).

1. Instalación y Configuración

Para comenzar a trabajar con Swift, necesitas tener instalado Xcode, el entorno de desarrollo integrado (IDE) de Apple. Puedes descargarlo desde la App Store de macOS.

Pasos para instalar Xcode:

1. Abre la **App Store** en tu Mac.
2. Busca **Xcode**.
3. Haz clic en 'Instalar' y espera a que se complete.

2. Primeros Pasos con Swift

Hello World

El primer programa clásico en cualquier lenguaje es imprimir 'Hola Mundo'. En Swift, puedes hacerlo de la siguiente manera:

```
import Swift
print("¡Hola Mundo!")
```

- `import Swift` importa el módulo necesario para usar las funcionalidades del lenguaje.
- `print()` es una función que muestra texto en la consola.

3. Tipos de Datos

Swift es un lenguaje con tipado fuerte y estático, lo que significa que debes declarar qué tipo de datos estás utilizando.

Tipos básicos:

- **Int**: Números enteros.

- **Double**: Números con decimales de precisión doble.
- **String**: Cadenas de texto.
- **Bool**: Valores booleanos (true o false).

Ejemplo:

```
var edad: Int = 25
var altura: Double = 1.75
var nombre: String = "Juan"
var esMayorDeEdad: Bool = true
```

Conversión entre tipos:

```
let numeroString = "42"
let numeroInt = Int(numeroString) // Convierte a tipo entero
```

4. Variables y Constantes

En Swift, las variables son declaradas con `var` y las constantes con `let`. Las constantes no pueden cambiar su valor una vez que se asigna.

Ejemplo:

```
var edad = 25
edad = 26 // Se puede modificar

let nombre = "Juan"
// nombre = "Pedro" // Error: no se puede modificar una constante
```

5. Operadores

Swift soporta los operadores comunes para realizar operaciones aritméticas, de comparación y lógicas.

Operadores Aritméticos:

```
let suma = 5 + 3
let resta = 5 - 3
let multiplicacion = 5 * 3
let division = 5 / 3
let modulo = 5 % 3
```

Operadores de Comparación:

```
let esIgual = (5 == 5)   // true
let esDiferente = (5 != 3) // true
let esMayorQue = (5 > 3)  // true
let esMenorQue = (5 < 3)  // false
```

Operadores Lógicos:

```
let esVerdadero = true && false // false
let esFalso = true || false // true
let noEsVerdadero = !true // false
```

6. Condicionales

Las estructuras condicionales se utilizan para ejecutar bloques de código dependiendo de condiciones.

if, else y else if:

```
let edad = 18

if edad >= 18 {
    print("Eres mayor de edad.")
} else {
    print("Eres menor de edad.")
}
```

switch:

```
let dia = "Lunes"

switch dia {
case "Lunes":
    print("Inicio de la semana.")
case "Viernes":
    print("Casi fin de semana.")
default:
    print("Un día común.")
}
```

7. Funciones

Las funciones son bloques de código reutilizables. En Swift, se definen con la palabra clave `func`.

Ejemplo:

```
func saludar(nombre: String) -> String {
    return "Hola, \(nombre)!"
}

let mensaje = saludar(nombre: "Juan")
print(mensaje)
```

8. Arreglos (Arrays)

Un arreglo es una colección ordenada de elementos.

Definición y uso:

```
var numeros: [Int] = [1, 2, 3, 4]
numeros.append(5) // Añadir un elemento
print(numeros[0]) // Imprime el primer elemento
```

Operaciones comunes sobre arreglos:

- Acceder a un elemento: `numeros[índice]`
- Añadir un elemento: `numeros.append(elemento)`
- Modificar un elemento: `numeros[índice] = valor`
- Eliminar un elemento: `numeros.remove(at: índice)`
- Contar elementos: `numeros.count`
- Verificar si está vacío: `numeros.isEmpty`
- Ordenar: `numeros.sort()`

9. Diccionarios (Dictionaries)

Un diccionario es una colección no ordenada de pares clave-valor.

Ejemplo:

```
var persona: [String: String] = ["nombre": "Juan", "apellido": "Pérez"]
persona["edad"] = "25" // Añadir un nuevo par clave-valor
print(persona["nombre"]!) // Imprime "Juan"
```

Operaciones comunes sobre diccionarios:

- Acceder a un valor: `persona[clave]`
- Añadir un par clave-valor: `persona["clave"] = valor`
- Eliminar un par clave-valor: `persona.removeValue(forKey: clave)`

- Verificar si contiene una clave: `persona.keys.contains(clave)`
- Contar elementos: `persona.count`
- Verificar si está vacío: `persona.isEmpty`

10. Clases y Estructuras

Swift es un lenguaje orientado a objetos, y te permite definir tus propias clases y estructuras.

Clases:

```
class Persona {
    var nombre: String
    var edad: Int

    init(nombre: String, edad: Int) {
        self.nombre = nombre
        self.edad = edad
    }

    func describir() -> String {
        return "Mi nombre es \(nombre) y tengo \(edad) años."
    }
}

let juan = Persona(nombre: "Juan", edad: 25)
print(juan.describir())
```

Estructuras:

```
struct Auto {
    var marca: String
    var modelo: String

    func describir() -> String {
        return "Este es un \(marca) \(modelo)."
    }
}

let miAuto = Auto(marca: "Toyota", modelo: "Corolla")
print(miAuto.describir())
```

11. Opcionales (Optionals)

Los opcionales en Swift representan un valor que puede estar presente o ser `nil` (ausente).

Ejemplo:

```
var nombre: String? = "Juan"
print(nombre) // Imprime Optional("Juan")

nombre = nil
print(nombre) // Imprime nil
```

Puedes desempaquetar un opcional utilizando `if let` o `guard let`:

```
if let nombreDesempaquetado = nombre {
    print("El nombre es \(nombreDesempaquetado)")
} else {
    print("No hay nombre disponible.")
}
```

12. Herencia y Polimorfismo

Swift es un lenguaje orientado a objetos y soporta herencia y polimorfismo.

Ejemplo de Herencia

```
class Animal {
    func hacerSonido() {
        print("El animal hace un sonido")
    }
}

class Perro: Animal {
    override func hacerSonido() {
        print("El perro ladra")
    }
}

let miPerro = Perro()
miPerro.hacerSonido()  // Imprime "El perro ladra"
```

Polimorfismo

El polimorfismo permite que puedas tratar objetos de diferentes clases de manera uniforme, como si fueran del mismo tipo.

```
let miAnimal: Animal = Perro()
miAnimal.hacerSonido()  // Imprime "El perro ladra"
```

13. Closure con Parámetros y Retorno

```
let multiplicar = { (a: Int, b: Int) -> Int in
    return a * b
}

let resultado = multiplicar(5, 3)  // 15
```

Trailing Closure

Cuando pasas un closure como el último argumento de una función, puedes escribirlo fuera de los par

14. Guard

```swift
func checkNumber(_ number: Int?) {

    guard let number = number else {

        print("No hay número")

        return

    }

    print("El número es \(number)")

}
```