Referencia Rápida de Swift

Tipos, opcionales, protocolos, manejo de errores esenciales

Fundamentos

Hola Mundo

```
import Foundation
print("Hello, World!")
```

Constantes y Variables

```
let name = "Swift"       // constante (inmutable)
var count = 0            // variable (mutable)
count += 1
let pi: Double = 3.14   // anotación de tipo explícita
```

Comentarios

```
// comentario de una línea
/* comentario
   multilínea */
/// comentario de documentación (soporta Markdown)
```

Tipos

Tipos Básicos

`Int`	Entero del tamaño de la plataforma (64 bits en sistemas modernos)
`Double`	Punto flotante de 64 bits (preferido sobre Float)
`Float`	Punto flotante de 32 bits
`Bool`	**`true`** / **`false`**
`String`	Cadena Unicode, tipo de valor
`Character`	Único clúster de grafema extendido

Inferencia de Tipos y Conversión

```
let score = 95              // inferido como Int
let gpa = 3.8              // inferido como Double
let total = Double(score) + gpa  // conversión explícita
let label = "Score: \(score)"    // interpolación de cadena
```

Tuplas

```
let point = (x: 3, y: 5)
print(point.x)            // acceso con nombre
let (x, y) = point        // descomponer
let (first, _) = point    // ignorar segundo valor
```

Alias de Tipos

```
typealias Coordinate = (Double, Double)
let origin: Coordinate = (0.0, 0.0)
```

Control de Flujo

If / Else

```
if score > 90 { print("A") }
else if score > 80 { print("B") }
else { print("C") }
```

Switch

```
switch grade {
case "A": print("excellent")
case "B", "C": print("passing")
default: print("unknown")
}
```

Bucles

```
for i in 0..<5 { }            // rango semi-abierto
for name in names { }          // colección
for (i, val) in list.enumerated() { }
while condition { }
repeat { } while condition     // do-while
```

Guard

```
func process(value: Int?) {
    guard let v = value, v > 0 else { return }
    print(v)   // v está desenvuelto y en ámbito
}
```

Funciones

Función Básica

```
func greet(name: String) -> String {
    return "Hello, \(name)!"
}
greet(name: "Alice")
```

Etiquetas de Argumento

```
func move(from start: Int, to end: Int) -> Int {
    return end - start
}
move(from: 0, to: 10)  // etiquetas externas
func add(_ a: Int, _ b: Int) -> Int { a + b }
```

Parámetros por Defecto y Variádicos

```
func join(_ items: String..., separator: String = ", ") -> String {
    items.joined(separator: separator)
}
join("a", "b", "c")
```

Parámetros inout

```
func double(_ x: inout Int) { x *= 2 }
var num = 5
double(&num)   // num ahora es 10
```

Closures

Sintaxis de Closure

```
let double = { (x: Int) -> Int in return x * 2 }
let nums = [3, 1, 2]
let sorted = nums.sorted { $0 < $1 }
let mapped = nums.map { $0 * 10 }
```

Closure al Final

```
UIView.animate(withDuration: 0.3) {
    view.alpha = 0.0
}
```

Capturar Valores

```
func makeCounter() -> () -> Int {
    var count = 0
    return { count += 1; return count }
}
let counter = makeCounter()  // counter() => 1, 2, ...
```

Clases y Structs

Struct (Tipo de Valor)

```
struct Point {
    var x: Double
    var y: Double
}
var p = Point(x: 1, y: 2)  // inicializador memberwise automático
```

Class (Tipo de Referencia)

```
class Vehicle {
    var speed: Double = 0
    init(speed: Double) { self.speed = speed }
}
class Car: Vehicle { var gear: Int = 1 }
```

Struct vs Class

`struct`	Tipo de valor, copiado al asignar, sin herencia
`class`	Tipo de referencia, compartido por referencia, soporta herencia
`mutating`	Palabra clave obligatoria para métodos de struct que modifican **`self`**
`deinit`	Desinicializador solo para clases (llamado antes de desasignación)

Protocolos

Definir y Conformar

```
protocol Drawable {
    var description: String { get }
    func draw()
}
struct Circle: Drawable { /* implementar miembros requeridos */ }
```

Extensiones de Protocolo

```
extension Drawable {
    func log() { print("Drawing: \(description)") }
}
// todos los conformes a Drawable obtienen log() gratis
```

Protocolos Comunes

`Equatable`	Comparación con **`==`** y **`!=`**
`Comparable`	Ordenamiento con **`<`**, **`>`**, **`<=`**, **`>=`**
`Hashable`	Puede usarse como clave de Dictionary o en Set
`Codable`	Encodable + Decodable (JSON, Plist)
`CustomStringConvertible`	Propiedad **`description`** personalizada
`Identifiable`	Requiere propiedad **`id`** (SwiftUI)

Opcionales

Declarar Opcionales

```
var name: String? = "Alice"  // puede contener String o nil
var age: Int? = nil          // actualmente nil
let count: Int = 5           // no opcional, nunca nil
```

Desenvolver

```
if let n = name { print(n) }       // enlace opcional
guard let n = name else { return }  // guard
let n = name ?? "Unknown"           // coalescencia de nil
let n = name!                       // desenvoltura forzada (falla
si nil)
```

Encadenamiento Opcional

```
let count = user?.address?.zip?.count
// retorna nil si cualquier eslabón de la cadena es nil
user?.save()   // llamado solo si user no es nil
```

Map Opcional

```
let length = name.map { $0.count }   // Int?
let upper = name.flatMap { $0.isEmpty ? nil : $0.uppercased() }
```

Enums

Enum Básico

```
enum Direction {
    case north, south, east, west
}
var heading = Direction.north
heading = .east   // tipo inferido
```

Valores Asociados

```
enum Result {
    case success(data: String)
    case failure(code: Int, message: String)
}
if case .failure(let code, _) = r { print(code) }
```

Valores Raw

```
enum Planet: Int {
    case mercury = 1, venus, earth, mars
}
let p = Planet(rawValue: 3)   // Optional(.earth)
print(Planet.earth.rawValue)  // 3
```

Métodos en Enums

```
enum Suit: String, CaseIterable {
    case hearts, diamonds, clubs, spades
}
Suit.allCases.forEach { print($0.rawValue) }
```

Manejo de Errores

Definir Errores

```
enum NetworkError: Error {
    case badURL
    case timeout(seconds: Int)
    case serverError(code: Int)
}
```

Lanzar y Capturar

```
func fetch(url: String) throws -> Data {
    guard url.hasPrefix("https") else { throw NetworkError.badURL }
    return Data()
}
do { let data = try fetch(url: "https://example.com") }
catch { print("Error: \(error)") }
```

Variantes de try

`try`	Debe estar dentro de **`do-catch`**, propaga el error
`try?`	Devuelve opcional, **`nil`** en caso de error
`try!`	try forzado, falla en caso de error
`throws`	La función puede lanzar errores
`rethrows`	Lanza solo si el argumento de closure lanza