Referência Rápida de Swift

Tipos, opcionais, protocolos e tratamento de erros essenciais

Básico

Hello World

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

Constantes e Variáveis

```
let name = "Swift"       // constante (imutável)
var count = 0            // variável (mutável)
count += 1
let pi: Double = 3.14   // anotação de tipo explícita
```

Comentários

```
// comentário de linha única
/* comentário
   multi-linha */
/// comentário de documentação (Markdown suportado)
```

Tipos

Tipos Básicos

`Int`	Inteiro de tamanho de plataforma (64-bit em sistemas modernos)
`Double`	Ponto flutuante de 64 bits (preferível ao Float)
`Float`	Ponto flutuante de 32 bits
`Bool`	**`true`** / **`false`**
`String`	String Unicode, tipo de valor
`Character`	Cluster de grafema estendido único

Inferência de Tipo e Conversão

```
let score = 95              // inferido como Int
let gpa = 3.8              // inferido como Double
let total = Double(score) + gpa  // conversão explícita
let label = "Score: \(score)"    // interpolação de string
```

Tuplas

```
let point = (x: 3, y: 5)
print(point.x)            // acesso nomeado
let (x, y) = point        // decompor
let (first, _) = point    // ignorar segundo valor
```

Aliases de Tipo

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

Fluxo de Controle

If / Else

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

Switch

```
switch grade {
case "A": print("excelente")
case "B", "C": print("aprovado")
default: print("desconhecido")
}
```

Loops

```
for i in 0..<5 { }           // intervalo semi-aberto
for name in names { }         // coleção
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 desempacotado e no escopo
}
```

Funções

Função Básica

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

Rótulos de Argumento

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

Parâmetros Padrão e Variádicos

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

Parâmetros inout

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

Closures

Sintaxe 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 Trailing

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

Capturando Valores

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

Classes e Structs

Struct (Tipo de Valor)

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

Class (Tipo de Referência)

```
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 na atribuição, sem herança
`class`	Tipo de referência, compartilhado por referência, suporta herança
`mutating`	Palavra-chave obrigatória para métodos de struct que modificam **`self`**
`deinit`	Desinicializador exclusivo de class (chamado antes da desalocação)

Protocolos

Definindo e Conformando

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

Extensões de Protocolo

```
extension Drawable {
    func log() { print("Drawing: \(description)") }
}
// todos os conformantes de Drawable ganham log() gratuitamente
```

Protocolos Comuns

`Equatable`	Comparação com **`==`** e **`!=`**
`Comparable`	Ordenação com **`<`**, **`>`**, **`<=`**, **`>=`**
`Hashable`	Pode ser usado como chave de Dictionary ou em Set
`Codable`	Encodable + Decodable (JSON, Plist)
`CustomStringConvertible`	Propriedade **`description`** personalizada
`Identifiable`	Requer propriedade **`id`** (SwiftUI)

Opcionais

Declarando Opcionais

```
var name: String? = "Alice"  // pode conter String ou nil
var age: Int? = nil          // atualmente nil
let count: Int = 5           // não opcional, nunca nil
```

Desempacotando

```
if let n = name { print(n) }       // ligação opcional
guard let n = name else { return }  // guard
let n = name ?? "Unknown"           // coalescência de nil
let n = name!                       // desempacotamento forçado
(falha se nil)
```

Encadeamento Opcional

```
let count = user?.address?.zip?.count
// retorna nil se qualquer elo da cadeia for nil
user?.save()   // chamado apenas se user não for 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
```

Referência Rápida de Swift

Valores Associados

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

Valores Brutos

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

Métodos em Enums

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

Tratamento de Erros

Definindo Erros

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

Lançando e Capturando

```
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`	Deve estar dentro de **`do-catch`**, propaga erro
`try?`	Retorna opcional, **`nil`** em caso de erro
`try!`	Try forçado, falha em caso de erro
`throws`	Função pode lançar erros
`rethrows`	Lança apenas se o argumento de closure lançar