

Tham Khảo Nhanh Swift

Kiểu dữ liệu, optionals, protocols, xử lý lỗi

Cơ bản

Hello World

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

Hằng số & Biến số

```
let name = "Swift" // constant (immutable)
var count = 0 // variable (mutable)
count += 1
let pi: Double = 3.14 // explicit type annotation
```

Chú thích

```
// single-line comment
/* multi-line
comment */
/// documentation comment (Markdown supported)
```

Kiểu dữ liệu

Kiểu cơ bản

Int	Số nguyên theo nền tảng (64-bit trên hệ thống hiện đại)
Double	Số thực dấu phẩy động 64-bit (Ưu tiên hơn Float)
Float	Số thực dấu phẩy động 32-bit
Bool	true / false
String	Chuỗi Unicode, kiểu giá trị
Character	Một cụm grapheme mở rộng

Suy luận kiểu & Chuyển đổi

```
let score = 95 // inferred as Int
let gpa = 3.8 // inferred as Double
let total = Double(score) + gpa // explicit conversion
let label = "Score: \(score)" // string interpolation
```

Tuples

```
let point = (x: 3, y: 5)
print(point.x) // named access
let (x, y) = point // decompose
let (first, _) = point // ignore second value
```

Type Aliases

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

Luồng điều khiển

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")
}
```

Vòng lặp

```
for i in 0..<5 { // half-open range
for name in names { // collection
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 is unwrapped and in scope
}
```

Hàm

Hàm cơ bản

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

Nhãn tham số

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

Tham số mặc định & Variadic

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

Tham số inout

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

Closures

Cú pháp 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 }
```

Trailing Closure

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

Bắt giá trị

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

Classes & Structs

Struct (Kiểu giá trị)

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

Class (Kiểu tham chiếu)

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

Struct vs Class

struct	Kiểu giá trị, sao chép khi gán, không kế thừa
class	Kiểu tham chiếu, chia sẻ qua tham chiếu, hỗ trợ kế thừa
mutating	Từ khóa bắt buộc cho phương thức struct sửa đổi self
deinit	Chỉ có trong class, gọi trước khi giải phóng bộ nhớ

Protocols

Định nghĩa & Conform

```
protocol Drawable {
var description: String { get }
func draw()
}
struct Circle: Drawable { /* implement required members */ }
```

Protocol Extensions

```
extension Drawable {
func log() { print("Drawing: \(description)") }
}
// all Drawable conformers get log() for free
```

Protocols phổ biến

Equatable	So sánh == và !=
Comparable	Thứ tự <, >, <=, >=
Hashable	Dùng làm key Dictionary hoặc trong Set
Codable	Encodable + Decodable (JSON, plist)
CustomStringConvertible	Thuộc tính description tùy chỉnh
Identifiable	Yêu cầu thuộc tính id (SwiftUI)

Optionals

Khai báo Optionals

```
var name: String? = "Alice" // may contain String or nil
var age: Int? = nil // currently nil
let count: Int = 5 // non-optional, never nil
```

Unwrapping

```
if let n = name { print(n) } // optional binding
guard let n = name else { return } // guard
let n = name ?? "Unknown" // nil coalescing
let n = name! // force unwrap (crashes if nil)
```

Optional Chaining

```
let count = user?.address?.zip?.count
// returns nil if any link in the chain is nil
user?.save() // called only if user is non-nil
```

Optional Map

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

Enums

Enum cơ bản

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

Associated Values

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

Tham Khảo Nhanh Swift

Raw Values

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

Phương thức trong Enum

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

Xử lý lỗi

Định nghĩa lỗi

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

Throw & Catch

```
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)") }
```

Các biến thể try

try	Phải nằm trong do-catch , lan truyền lỗi
try?	Trả về optional, nil khi có lỗi
try!	Force try, crash khi có lỗi
throws	Hàm có thể throw lỗi
rethrows	Chỉ throw nếu closure tham số throw