Swift クイックリファレンス

型、オプショナル、プロトコル、エラーハンドリングの基本

基本

Hello World

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

定数と変数

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

コメント

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

型

基本型

Int	プラットフォームサイズの整数（現代システムでは 64 ビット）
Double	64 ビット浮動小数点（Float より推奨）
Float	32 ビット浮動小数点
Bool	**true** / **false**
String	Unicode 文字列、値型
Character	単一の拡張書記素クラスター

型推論と変換

```
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
```

タプル

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

型エイリアス

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

制御フロー

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")
}
```

ループ

```
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
}
```

関数

基本的な関数

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

引数ラベル

```
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 }
```

デフォルトと可変長パラメータ

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

inout パラメータ

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

クロージャ

クロージャ構文

```
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 }
```

末尾クロージャ

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

値のキャプチャ

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

クラスと構造体

struct（値型）

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

class（参照型）

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

struct vs class

struct	値型、代入時にコピー、継承なし
class	参照型、参照で共有、継承をサポート
mutating	**self** を変更する struct メソッドに必要なキーワード
deinit	クラス専用のデイニシャライザ（解放前に呼ばれる）

プロトコル

定義と準拠

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

プロトコル拡張

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

主要プロトコル

Equatable	**==** と **!=** の比較
Comparable	**<**、**>**、**<=**、**>=** の順序比較
Hashable	Dictionary のキーや Set に使用可能
Codable	Encodable + Decodable（JSON、Plist）
CustomStringConvertible	カスタム **description** プロパティ
Identifiable	**id** プロパティが必要（SwiftUI）

オプショナル

オプショナルの宣言

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

アンラップ

```
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)
```

オプショナルチェーン

```
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
```

オプショナルの map

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

Swift クイックリファレンス

列挙型

基本的な enum

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

関連値

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

生の値

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

enum のメソッド

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

エラーハンドリング

エラーの定義

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

スローとキャッチ

```
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)") }
```

try のバリエーション

try	**do-catch** 内で必須、エラーを伝播
try?	オプショナルを返す、エラー時は **nil**
try!	強制 try、エラー時にクラッシュ
throws	関数がエラーをスローできる
rethrows	クロージャ引数がスローした場合のみスロー