Swift Quick Reference

Types, optionals, protocols, error handling essentials

Basics

Hello World

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

Constants & Variables

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

Comments

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

Types

Basic Types

`Int`	Platform-sized integer (64-bit on modern systems)
`Double`	64-bit floating point (preferred over Float)
`Float`	32-bit floating point
`Bool`	**`true`** / **`false`**
`String`	Unicode string, value type
`Character`	Single extended grapheme cluster

Type Inference & Conversion

```
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)
```

Control Flow

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")
}
```

Loops

```
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
}
```

Functions

Basic Function

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

Argument Labels

```
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 }
```

Default & Variadic Parameters

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

inout Parameters

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

Closures

Closure Syntax

```
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
}
```

Capturing Values

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

Classes & Structs

Struct (Value Type)

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

Class (Reference Type)

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

Struct vs Class

`struct`	Value type, copied on assignment, no inheritance
`class`	Reference type, shared by reference, supports inheritance
`mutating`	Required keyword for struct methods that modify **`self`**
`deinit`	Class-only deinitializer (called before deallocation)

Protocols

Defining & Conforming

```
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
```

Common Protocols

`Equatable`	**`==`** and **`!=`** comparison
`Comparable`	**`<`**, **`>`**, **`<=`**, **`>=`** ordering
`Hashable`	Can be used as Dictionary key or in Set
`Codable`	Encodable + Decodable (JSON, Plist)
`CustomStringConvertible`	Custom **`description`** property
`Identifiable`	Requires **`id`** property (SwiftUI)

Optionals

Declaring 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

Basic Enum

```
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) }
```

Swift Quick Reference

Raw Values

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

Methods on Enums

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

Error Handling

Defining Errors

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

Throwing & Catching

```
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 Variants

`try`	Must be inside **`do-catch`**, propagates error
`try?`	Returns optional, **`nil`** on error
`try!`	Force try, crashes on error
`throws`	Function can throw errors
`rethrows`	Throws only if closure argument throws