RÉFÉRENCE RAPIDE SWIFT

Types, optionnels, protocoles, gestion des erreurs

Bases

Bonjour le monde

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

Constantes et variables

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

Commentaires

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

Types

Types de base

`Int`	Entier de taille native (64 bits sur les systèmes modernes)
`Double`	Virgule flottante 64 bits (préféré à Float)
`Float`	Virgule flottante 32 bits
`Bool`	`true` / `false`
`String`	Chaîne Unicode, type valeur
`Character`	Seul groupe de graphèmes étendu

Inférence de type et 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
```

Alias de type

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

Flux de contrôle

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")
}
```

Boucles

```
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
}
```

Fonctions

Fonction de base

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

Étiquettes d'argument

```
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 }
```

Paramètres par défaut et variadiques

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

Paramètres inout

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

Fermetures

Syntaxe de fermeture

```
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 }
```

Fermeture de fin

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

Capture de valeurs

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

Classes et structures

Struct (type valeur)

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

Classe (type référence)

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

Struct vs Classe

`struct`	Type valeur, copié à l'affectation, sans héritage
`class`	Type référence, partagé par référence, supporte l'héritage
`mutating`	Mot-clé requis pour les méthodes de struct qui modifient `self`
`deinit`	Deinitializer réservé aux classes (appelé avant la libération)

Protocoles

Définir et se conformer

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

Extensions de protocole

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

Protocoles courants

`Equatable`	Comparaison `==` et `!=`
`Comparable`	Ordre `<`, `>`, `<=`, `>=`
`Hashable`	Peut être utilisé comme clé de Dictionary ou dans un Set
`Codable`	Encodable + Decodable (JSON, Plist)
`CustomStringConvertible`	Propriété `description` personnalisée
`Identifiable`	Requiert une propriété `id` (SwiftUI)

Optionnels

Déclarer des optionnels

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

Déballage

```
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)
```

Chaînage optionnel

```
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
```

Map optionnel

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

Énumérations

Énumération de base

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

Valeurs associées

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

Valeurs brutes

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

Méthodes sur les énumérations

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

Gestion des erreurs

Définir des erreurs

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

Lancer et attraper

```
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`	Doit être dans un `do-catch`, propage l'erreur
`try?`	Retourne un optionnel, `nil` en cas d'erreur
`try!`	Force, plante en cas d'erreur
`throws`	La fonction peut lancer des erreurs
`rethrows`	Lance seulement si l'argument closure lance