Swift Compiled language and SwiftPM

START HERE

`swift --version`
Check the toolchain

`swift package init --type executable`
New CLI project

`swift build`
Compile

`swift run`
Build and run

`swift test`
Run tests

`swift repl`
Interactive prompt

`swiftc <file>.swift -o <out>`
Compile one file

`swift package resolve`
Fetch dependencies

Package.swift manifest → `swift package resolve` → **Sources/** targets → `swift build` → **.build/** binaries → `swift run` → **Output** your program

CONCEPTS

Swift Package Manager (swift package, build, run, test) is built in; Package.swift is the manifest.

Prefer let over var; the compiler warns when a var is never mutated.

TOOLCHAIN

`swift --version`
Show Swift version

`swift <file>.swift`
Run a script directly

`swiftc <file>.swift -o <out>`
Compile to an executable

`swiftc -O <file>.swift`
Optimized build

`swift repl`
Start the REPL

`swift run <target> <args>`
Run a named executable with args

`swift build -c release`
Optimized release build

`swift build --show-bin-path`
Print the output directory

`swift test --filter <name>`
Run matching tests only

`swift test --parallel`
Run tests in parallel

PACKAGE MANAGER

`swift package init --name <name> --type executable`
Create an executable package

`swift package init --type library`
Create a library package

`swift package resolve`
Resolve and fetch dependencies

`swift package update`
Update to newest allowed versions

`swift package clean`
Remove build artifacts

`swift package show-dependencies`
Print the dependency tree

`.package(url: "<url>", from: "1.0.0")`
Declare a dependency

`.executableTarget(name: "<name>")`
Declare an executable target

PACKAGE MANAGER (cont.)

`.testTarget(name: "<name>Tests", dependencies: ["<name>"])`
Declare a test target

`// swift-tools-version: 6.0`
First line of Package.swift

BASICS

`let x = 1`
Constant

`var x = 1`
Variable

`let x: Double = 1`
Explicit type

`"Hi \(name)"`
String interpolation

`let (a, b) = (1, 2)`
Destructure a tuple

`let r = 1...5`
Closed range

`1..<5`
Half-open range

`for i in 0..<n { }`
Loop over a range

`while <cond> { }`
While loop

`if <cond> { } else { }`
Conditional

`switch x { case 1: ... default: ... }`
Pattern match, must be exhaustive

`func f(_ a: Int, b: Int = 0) → Int { }`
Function with labels

`f(1, b: 2)`
Call with argument label

`print(x)`
Print a value

OPTIONALS

`var x: Int? = nil`
Optional value

`if let v = x { }`
Unwrap if present

`guard let v = x else { return }`
Early exit unwrap

`x ?? <default>`
Fallback value

`x?.count`
Optional chaining

`x!`
Force unwrap (crashes on nil)

OPTIONALS (cont.)

`if let x { }`
Shorthand unwrap

COLLECTIONS

`let a = [1, 2, 3]`
Array literal

`a.append(4)`
Add an element

`a[0]`
Subscript

`a.count / a.isEmpty`
Size checks

`let d = ["k": 1]`
Dictionary literal

`d["k"]`
Lookup returns an optional

`let s: Set = [1, 2]`
Set

`a.map { $0 * 2 }`
Transform elements

`a.filter { $0 > 1 }`
Keep matches

`a.reduce(0, +)`
Fold to a value

`a.sorted()`
Sorted copy

`a.compactMap { Int($0) }`
Map and drop nils

`for (i, v) in a.enumerated() { }`
Index and value

TYPES

`struct P { var x: Int }`
Value type

`class C { var x = 0 }`
Reference type

`enum D { case n, s }`
Enumeration

`case some(Int)`
Enum case with payload

`protocol Shape { func area() → Double }`
Protocol

`extension P: Shape { }`
Conform in an extension

`init(x: Int) { self.x = x }`
Initializer

`mutating func inc() { }`
Mutate a struct

`var area: Double { w * h }`
Computed property

`struct Box<T> { var v: T }`
Generic type

TYPES (cont.)

`func f<T: Comparable>(_ a: T) { }`
Generic function

`{ (a: Int) → Int in a * 2 }`
Closure

ERRORS AND ASYNC

`func f() throws → Int`
Function that can throw

`do { try f() } catch { }`
Handle an error

`try? f()`
Optional result on failure

`throw MyError.bad`
Throw an error

`enum MyError: Error { case bad }`
Define an error type

`defer { <cleanup> }`
Run on scope exit

`func f() async → Int`
Async function

`await f()`
Call an async function

`Task { await f() }`
Start async work

`async let a = f()`
Run child tasks in parallel

`withTaskGroup(of: Int.self) { g in }`
Dynamic task group

`actor Counter { }`
Isolate mutable state

`@main struct App { static func main() }`
Program entry point

RECIPES

New CLI project

```
swift package init --name hello
--type executable
cd hello
swift run
```

Add a dependency

```
# Package.swift
dependencies: [
  .package(url: "https://
github.com/apple/swift-
argument-parser", from: "1.3.
0")
],
# then: swift build
```

Release build and test

```
swift test
swift build -c release
.build/release/<name>
```

Quick script

```
echo 'print("hi")' > t.swift
swift t.swift
```