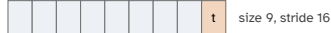


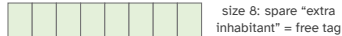
In one line: `T?` is sugar for the stdlib `enum Optional<Wrapped> { case none; case some(Wrapped) }`; `nil` is `.none` via `ExpressibleByNilLiteral`. Absence is part of the **type**, so the compiler forces you to unwrap — binding, chaining, `??`, `map`, pattern matching — and `!` is an explicit bet that traps on `nil`.

A it is an enum: tag + payload

`Int?` = 8-byte payload + 1 tag byte

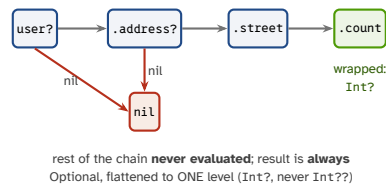


`UIView?` = one pointer, `.none` = `0x0`

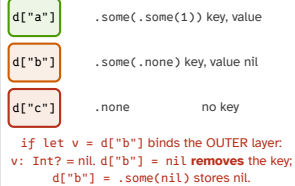


`MemoryLayout<Int?>.size == 9` · Optional of a class costs 0 bytes

B chaining short-circuits



C `[String: Int?] lookup = Int??`



How it works

- Binding:** `if let x = opt, guard let x = opt else { return }` (`x` stays in scope *after* the guard). Swift 5.7 shorthand (SE-0345): `if let x {} / guard let x else {} = if let x = x`. Binds an immutable non-optional; `if var` for a mutable copy.
- Chaining** `a?.b?.c()`: first `nil` stops the rest (incl. argument evaluation); the result is `T?`. A `Void` method gives `Void?` — compare to `nil` to know if it ran; `a?.x = 5` also returns `Void?`.
- ??:** `func ??<T>(T?, @autoclosure () throws -> T) rethrows -> T` (+ an overload returning `T?`). Right side evaluated **only if nil**; right-associative: `a ?? b ?? c = a ?? (b ?? c)`.
- !** traps: “Fatal error: Unexpectedly found nil while unwrapping an Optional value” (EXC_BREAKPOINT on arm64). Only for invariants: a literal `URL(string:!)`, a bundled resource.
- IUO T!** (SE-0054, Swift 4.2) is **not a type**: it is `Optional<T>` + an attribute “force at use if needed”. `let y = iuo infers y: T?`; only a use needing `T` forces. For late-set-but-always-there values: `@IBOutlet`, `UIKit` window.
- map** (`T`) $\rightarrow$ `U` $\rightarrow$ `U?`; **flatMap** (`T`) $\rightarrow$ `U?` $\rightarrow$ `U?` (no `U??`). `s.map { Int($0) }` is `Int??`; `flatMap` gives `Int?`.
- try?** flattens since Swift 5 (SE-0230): `try?` on throws $\rightarrow$ `Int?` is `Int?` (Swift 4: `Int??`).
- Conformances:** `Optional` is `Equatable/Hashable/Codable` when `Wrapped` is — but **never Comparable**: `no < on Int?`.

Example — every unwrap

```
func greet(_ name: String?) -> String {
    guard let name else { return "anon" } // 5.7 shorthand
    return "hi " + name                  // name: String
}
let n: Int? = Int("42")                 // failable init
let a = n ?? 0                          // Int
let b = n.map { $0 * 2 }                 // Int?
switch n {
case .some(let x) where x > 0: print(x) // enum case pattern
case let x?: print("<= 0", x)           // x? == .some(x)
case nil: print("none")                 // == .none
}
for case let id? in [1, nil, 3] { }      // skips nils
if case .some(let x) = n { print(x) }
```

Optional ≠ ownership

- `var d: Delegate?` is a **strong** reference — “optional” says *may be absent*, not *does not retain*.
- `weak var must` be `Optional` (`var`) because ARC sets it to `nil` on dealloc; `unowned` may be optional (Swift 5) but is never nilled — access after dealloc traps.
- `Optional` models *absence*; `throws/Result` model *failure with a reason*. Don’t smuggle errors into `nil`.

Optionals meet the APIs

- `compactMap` drops `nil`s: `["1", "x"].compactMap(Int.init) -> [1]` (map would give `[Int?]`).
- Lookups return optionals: `first(where:)`, `firstIndex(of:)`, `dict[k]`, `Array.first/last`; `dict[k, default: 0] += 1` avoids the unwrap.
- Synthesized `Codable`: an optional property uses `decodeIfPresent` (missing key $\rightarrow$ `nil`) and `encodeIfPresent` (`nil` $\rightarrow$ key **omitted**, not `null`).
- `"\{opt}"` prints `Optional(5)` + a compiler warning — unwrap or use `String(describing:)` knowingly.

Interview traps

- `d["k"]?.count` is `Int?`, not `Int??` — chaining flattens. But a `[K: V?]` subscript IS `V??` (panel C).
- `if x != nil { use(x!) }` — a check the compiler cannot see; someone removes the check later. Bind instead: `if let x`.
- `x ?? log()` — `log()` does **not** run when `x` is non-`nil` (autoclosure). Don’t hide side effects on the right.
- `x ?? 0 ?? 5` parses `x ?? (0 ?? 5)`: warning, left side non-optional, the 5 is never used.
- IUO propagation: `let y = outlet is T?` — the “!” is not inherited (since 4.2).
- `try?` hides which: “threw” and “returned `nil`” both become `nil`.
- `Bool?` in `if`: `if flag` does not compile; write `flag == true` (`nil` $\rightarrow$ `false`) deliberately.

Remember

“Optional is a box with two cases: bind it, chain it, default it, map it — bang it only when a crash is the right answer.”

Likely questions

- What is an `Optional`? — `enum Optional<Wrapped>, .none/.some`.
- `if let` vs `guard let`? — `guard`: early exit, value in scope after.
- Is `Int!` a type? — no, `Optional` + `implicit-unwrap` attribute (4.2).
- `map` vs `flatMap`? — `flatMap` when the closure returns `U?`.
- When does `T??` appear? — `[K: V?]` subscript, `map` with an optional closure.
- Why is `UIView?` free? — the null pointer is a spare bit pattern for `.none`.