Junior iOS Developer Swift Interview Questions (2025 Edition)

---

1. What are optionals in Swift? How do you handle them safely?

Optionals are a Swift feature that allows a variable to have a value or be `nil`.
Example:

```
var name: String? = "John"  // Optional string
name = nil  // Now it has no value
```

Ways to handle optionals safely:

Optional Binding (`if let`)

```
 if let unwrappedName = name {
    print(unwrappedName)
}
```

- **Guard Statement (`guard let`)**

```
func greet(_ name: String?) {

    guard let name = name else {
        print("No name provided")
        return
    }
    print("Hello, \(name)")
}
```

- **Nil-Coalescing (`??`)**

```
    let user = name ?? "Guest"

print(user)  // Prints "Guest" if name is nil
```

Created By : Ruchit B Shah
(Senior Principal Software Engineer - Mobile Developer - + 91 9228877722)

2. What is the difference between `var` and `let`?

- `var` (mutable) allows values to change.
- `let` (immutable) makes values constant.
  Example:

```
var age = 25
age = 30  // Allowed

let name = "John"
name = "Doe"  // Error: Cannot assign to 'let' constant
```

3. What is the difference between `struct` and `class` in Swift?

Feature	`struct` (Value Type)	`class` (Reference Type)
Memory Storage	Stack	Heap
Inheritance	❌ Not supported	✅ Supported
Mutability	Immutable by default	Mutable
Copy Behavior	Copies the value	Copies the reference
Performance	Faster	Slower due to ARC

Example:

```
struct Person {
    var name: String
}
var person1 = Person(name: "Alice")
var person2 = person1
person2.name = "Bob"

print(person1.name)  // "Alice" (unchanged)
print(person2.name)  // "Bob" (modified)
```

---

4. What are computed properties, and how are they different from stored properties?

- **Stored Property**: Stores a value.
- **Computed Property**: Dynamically calculates a value.

Example:

```
struct Rectangle {
    var width: Double
    var height: Double

    // Computed Property
    var area: Double {
        return width * height
    }
}

let rect = Rectangle(width: 10, height: 5)
print(rect.area)  // 50
```

---

5. What is type inference in Swift?

Swift automatically determines the data type.
Example:

```
let x = 10  // Swift infers x as Int
let y = "Hello"  // Swift infers y as String
```

---

6. What is the difference between `mutating` and `non-mutating` functions?

- `mutating` is needed when modifying properties inside a `struct`.
  Example:

```
struct Car {
    var speed = 0
    mutating func accelerate() {
        speed += 10
    }
}
```

- Classes don't require `mutating` since they are reference types.

---

7. What are extensions in Swift, and how do they work?

Extensions add functionality to existing types.
Example:

```
extension Int {
    func square() -> Int {
        return self * self
    }
}
print(4.square())  // 16
```

---

8. How do you define and use a protocol in Swift?

A protocol defines a blueprint of methods and properties.

```
protocol Vehicle {
    var speed: Int { get set }
    func accelerate()
}
struct Car: Vehicle {
    var speed = 0
    func accelerate() {
        print("Accelerating...")
    }
}
```

9. What is `Codable`, and how does it work in JSON parsing?

`Codable` helps in encoding and decoding JSON.
Example:

```
struct User: Codable {
    var name: String
    var age: Int
}
let json = """
{"name": "Alice", "age": 25}
""".data(using: .utf8)!

let decoder = JSONDecoder()
let user = try? decoder.decode(User.self, from: json)
print(user?.name)  // "Alice"
```

10. Explain Swift's access control levels (`open`, `public`, `internal`, `fileprivate`, `private`).

Modifier	Accessibility
`open`	Accessible anywhere, subclassable outside the module
`public`	Accessible anywhere but not subclassable outside the module
`internal`	Default, accessible within the same module
`fileprivate`	Accessible within the same file
`private`	Accessible within the same scope

Example:

```
class MyClass {
    private var secret = "Hidden"
}
```

11. What is protocol-oriented programming (POP) in Swift?

Swift encourages protocols instead of inheritance.

Example:

```
protocol Drivable {
    func drive()
}
struct Car: Drivable {
    func drive() {
        print("Driving")
    }
}
```

12. What is the difference between delegation and notification?

Feature	Delegation	Notification
One-to-One	✅	❌
One-to-Many	❌	✅
Uses	`protocol`	`NotificationCenter`

Example:

```
protocol CarDelegate {
    func didStartDriving()
}
```

13. What is the difference between `class` inheritance and protocol conformance?

- A class can inherit only **one superclass**, but conform to **multiple protocols**.
- Protocols define behavior without implementation.

Example:

```
class Animal { }  // Superclass
protocol Walkable { }
class Dog: Animal, Walkable { }  // Dog inherits from Animal and
conforms to Walkable
```

---

14. How does Swift handle multiple inheritance?

Swift **does not support** multiple inheritance. Instead, use **protocols**.

Example:

```
protocol Flyable { }
protocol Swimmable { }
class Bird: Flyable, Swimmable { }  // Supports both behaviors
```

---

15. What are associated types in Swift protocols?

Allows protocols to define a placeholder type.
Example:

```
protocol Container {
    associatedtype Item
    func add(item: Item)
}
```

---

16. What is a generic in Swift, and why is it useful?

Generics allow **flexible and reusable code**.
Example:

```
func swapValues<T>(a: inout T, b: inout T) {
    let temp = a
    a = b
    b = temp
}
```

17. Explain the SOLID principles in Swift.

SOLID principles help in writing maintainable and scalable code:

1. **S** - **Single Responsibility Principle**: A class should have only one reason to change.
2. **O** - **Open-Closed Principle**: Code should be open for extension but closed for modification.
3. **L** - **Liskov Substitution Principle**: Subtypes should be replaceable for their base types.
4. **I** - **Interface Segregation Principle**: A class shouldn't implement unnecessary methods it doesn't use.
5. **D** - **Dependency Inversion Principle**: High-level modules should not depend on low-level modules.

---

18. What is Automatic Reference Counting (ARC) in Swift?

ARC automatically manages memory by tracking strong references to objects. When no references remain, the object is deallocated.

Example:

```
class Person {
    var name: String
    init(name: String) {
        self.name = name
    }
}
```

ARC automatically frees memory when no instances are used.

---

19. What is the difference between strong, weak, and unowned references?

Reference Type	Ownership	Prevents Deallocation?	Use Case

`strong`	Default	Yes	Keeps object in memory
`weak`	No ownership	No	Avoids retain cycles (e.g., delegates)
`unowned`	No ownership	No	Used when the reference should never be nil

Example:

```swift
class Person {
    var pet: Pet?
}
class Pet {
    weak var owner: Person?  // Avoids retain cycle
}
```

---

20. What is a retain cycle, and how do you prevent it?

A retain cycle happens when two objects hold strong references to each other, preventing deallocation.
Solution: Use `weak` or `unowned` references.

Example of **Retain Cycle**:

```swift
class A {
    var b: B?
}
class B {
    var a: A?  // Retain cycle
}
```

Fix with **weak reference**:

```swift
class B {
    weak var a: A?
}
```

---

21. What is Grand Central Dispatch (GCD), and how do you use it?

GCD is Apple's API for concurrent execution.

Example:

swift
CopyEdit

```
DispatchQueue.global(qos: .background).async {
    print("Background Task")
    DispatchQueue.main.async {
        print("Back to Main Thread")
    }
}
```

---

22. What is an operation queue in iOS?

`OperationQueue` is a higher-level abstraction over GCD for better dependency management.

Example:

```
let queue = OperationQueue()
queue.addOperation {
    print("Operation Executed")
}
```

---

23. What are `async/await` in Swift, and how do they improve concurrency handling?

`async/await` simplifies asynchronous code execution.

Example:

```
func fetchData() async -> String {
    return "Data Received"
}

Task {
    let result = await fetchData()
    print(result)
}
```

---

24. What is `DispatchGroup`, and when do you use it?

`DispatchGroup` is used to synchronize multiple tasks.

Example:

```
let group = DispatchGroup()
group.enter()
DispatchQueue.global().async {
    print("Task 1")
    group.leave()
}
group.notify(queue: .main) {
    print("All tasks completed")
}
```

---

25. What is Auto Layout, and why is it important?

Auto Layout is a constraint-based layout system that makes UI adaptive to different screen sizes.

Example:

```
view.translatesAutoresizingMaskIntoConstraints = false
NSLayoutConstraint.activate([
    view.leadingAnchor.constraint(equalTo: superview.leadingAnchor),
    view.trailingAnchor.constraint(equalTo: superview.trailingAnchor)
])
```

---

26. What is the difference between `frame` and `bounds`?

Property	Definition
`frame`	Position and size **relative to superview**
`bounds`	Position and size **relative to itself**

Example:

```
print(view.frame)  // (x:10, y:10, width:200, height:200)
print(view.bounds) // (x:0, y:0, width:200, height:200)
```

---

27. What is the difference between `XIB`, `Storyboard`, and programmatic UI?

UI Method	Pros	Cons
Storyboard	Visual, drag-and-drop UI	Hard to manage in large projects
XIB	Reusable UI components	Not ideal for full apps
Programmatic UI	Full control	More code to write

Example of **Programmatic UI**:

```
let label = UILabel()
label.text = "Hello"
view.addSubview(label)
```

---

28. What is `UIViewController` lifecycle, and when do the methods get called?

Lifecycle Method	When It Runs
`viewDidLoad`	Once, when the view is created
`viewWillAppear`	Before the view appears
`viewDidAppear`	After the view appears

`viewWillDisappear`	Before the view disappears
`viewDidDisappear`	After the view disappears

Example:

```
override func viewDidLoad() {
    super.viewDidLoad()
    print("View Loaded")
}
```

---

29. What is the difference between `UITableView` and `UICollectionView`?

Component	Use Case
`UITableView`	Single-column list
`UICollectionView`	Grid-based layouts

Example:

```
let tableView = UITableView()
tableView.register(UITableViewCell.self, forCellReuseIdentifier:
"cell")
```

---

30. How do you pass data between view controllers?

1. **Using Segue (`prepareForSegue`)**

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    let destination = segue.destination as! SecondVC
    destination.data = "Hello"
}
```

2. **Using Delegation**

```
protocol DataPassingDelegate {
    func sendData(_ data: String)
}
```

3. **Using Closures**

```
class SecondVC {
    var completion: ((String) -> Void)?
}
```

4. **Using NotificationCenter**

```
NotificationCenter.default.post(name:
NSNotification.Name("DataReceived"), object: "Hello")
```

---

31. What is a diffable data source, and how does it improve table views?

Diffable data sources optimize UI updates by automatically managing changes.

Example:

```
var snapshot = NSDiffableDataSourceSnapshot<Section, Item>()
snapshot.appendSections([.main])
snapshot.appendItems(["Item1", "Item2"])
dataSource.apply(snapshot)
```

---

32. What is URLSession, and how do you use it to make network requests?

`URLSession` is the primary API for making network calls in Swift.

Example of a **GET request**:

```
let url = URL(string: "https://api.example.com/data")!

let task = URLSession.shared.dataTask(with: url) { data, response,
error in
```

```
    guard let data = data, error == nil else {
        print("Error:", error ?? "Unknown error")
        return
    }
    print("Data received:", String(data: data, encoding: .utf8)!)
}
task.resume()
```

---

33. How do you handle JSON parsing in Swift?

Use `Codable` for easy encoding/decoding of JSON data.

Example:

```
struct User: Codable {
    let name: String
    let age: Int
}

let jsonData = """
{"name": "Alice", "age": 25}
""".data(using: .utf8)!

let user = try? JSONDecoder().decode(User.self, from: jsonData)
print(user?.name)  // "Alice"
```

---

34. What is the difference between `URLSession` and third-party networking libraries like `Alamofire`?

Feature	`URLSession`	`Alamofire`
Complexity	More manual setup	Easier, less boilerplate
Dependencies	No external dependency	Requires `Alamofire`
Flexibility	Full control	High-level abstraction

Example using **Alamofire**:

```
import Alamofire

AF.request("https://api.example.com/data").responseJSON { response in
    print(response)
}
```

---

35. What is Combine, and how does it improve data handling?

Combine is Apple's reactive framework for handling asynchronous events like API responses.

Example of using **Combine for API requests**:

```
import Combine

struct User: Codable {
    let name: String
}

let url = URL(string: "https://api.example.com/user")!
let publisher = URLSession.shared.dataTaskPublisher(for: url)
    .map { $0.data }
    .decode(type: User.self, decoder: JSONDecoder())
    .sink(receiveCompletion: { print($0) },
          receiveValue: { print($0.name) })
```

---

36. What is UserDefaults, and when should you use it?

`UserDefaults` is used for storing small key-value pairs (e.g., user preferences).

Example:

```
UserDefaults.standard.set("Alice", forKey: "username")
let username = UserDefaults.standard.string(forKey: "username")
print(username)  // "Alice"
```

🚨 **Not recommended for large or sensitive data**.

---

37. What is Core Data, and how does it work?

Core Data is Apple's framework for managing object graphs and persisting data.

Example of saving data in **Core Data**:

```
let context = (UIApplication.shared.delegate as!
AppDelegate).persistentContainer.viewContext

let newUser = UserEntity(context: context)
newUser.name = "Alice"
newUser.age = 25

try? context.save()
```

---

38. What is the difference between Core Data and Realm?

Feature	Core Data	Realm
Performance	Slower	Faster
Setup Complexity	More setup required	Easier setup
Syncing	No built-in sync	Built-in sync

Example of **saving data in Realm**:

```
let realm = try! Realm()
let user = User()
user.name = "Alice"

try! realm.write {
    realm.add(user)
}
```

---

39. What is the MVC architecture?

Component	Responsibility
Model	Business logic, data storage
View	UI (e.g., labels, buttons)
Controller	Handles user input and updates the view

Example:

```
class User {
    var name: String
    init(name: String) { self.name = name }
}

class ViewController: UIViewController {
    var user = User(name: "Alice")
    override func viewDidLoad() {
        super.viewDidLoad()
        print(user.name)
    }
}
```

---

40. What is the MVVM architecture? How does it improve MVC?

MVVM introduces a `ViewModel`, which separates UI logic from business logic.

Component	Responsibility
Model	Data & business logic
ViewModel	Processes data for the View
View	Displays UI

Example:

```
class UserViewModel {
    var username: String
    init(user: User) {
        self.username = user.name.uppercased()
    }
}
```

---

41. What is the Singleton pattern?

A singleton ensures **only one instance** of a class exists.

Example:

```
class APIService {
    static let shared = APIService()
    private init() {}  // Prevents instantiation
}
```

---

42. How do you debug memory leaks in an iOS app?

- Use **Instruments > Leaks** tool
- Check for **retain cycles** using **Xcode Memory Graph**

---

43. What are breakpoints, and how do you use them?

Breakpoints pause execution for debugging.

- Add a **symbolic breakpoint** (e.g., on `viewDidLoad`)
- Use **LLDB commands**:

  ```
  po self  // Print object
  ```

---

44. How do you securely store sensitive user data?

Use **Keychain**, not `UserDefaults`.

Example:

```
let password = "secret"
let keychain = KeychainSwift()
keychain.set(password, forKey: "userPassword")
```

---

45. How do you improve app launch time?

- Minimize heavy operations in `viewDidLoad`
- Use background threads (`DispatchQueue.global`)
- Optimize images using `Image Assets`

---

46. How do you handle deep linking in iOS?

Deep linking allows opening specific app pages via URLs.

Example:

```
func application(_ app: UIApplication, open url: URL) -> Bool {
    print("Opened via \(url)")
    return true
}
```

---

47. What is App Transport Security (ATS)?

ATS forces apps to use **HTTPS** connections for security.

To disable ATS (not recommended):

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

---

48. What are Unit Tests and UI Tests in iOS?

Test Type	Purpose
Unit Test	Tests individual methods
UI Test	Simulates user interactions

Example of a **unit test**:

```
func testExample() {
    let sum = add(2, 3)
    XCTAssertEqual(sum, 5)
}
```

---

49. How do you distribute an iOS app?

1. **App Store** (via App Store Connect)
2. **TestFlight** (for beta testing)
3. **Enterprise Distribution** (for internal apps)

---

50. How do you handle app crashes in production?

- Use **Crashlytics** for logging errors
- Implement **NSExceptionHandler**
- Collect user crash reports

Created By : Ruchit B Shah
(Senior Principal Software Engineer - Mobile Developer - + 91 9228877722)