Preguntas de la entrevista de Swift para desarrolladores junior de iOS (edición 2025)

1. **¿Qué son los opcionales en Swift? ¿Cómo los manejas de manera segura?**

Los opcionales son una característica de Swift que permite que una variable tenga un valor o sea nula.

Ejemplo:

```
var name: String? = "John" // Optional string
name = nil // Ahora no tiene valor
```

Formas de manejar los opcionales de forma segura:

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" // Si el name es nulo
```

2. **¿Cuál es la diferencia entre var y let?**

var (mutable) permite que los valores cambien.
let (inmutable) hace que los valores sean constantes.

Ejemplo:

```
var age = 25 age = 30 // Allowed
let name = "John"
name = "Doe" // Error: No se puede asignar a la constante 'let'
```

3. **¿Cuál es la diferencia entre struct y class en Swift?**

Característica	struct (tipo de valor)	class (Tipo de referencia)
Almacenamiento de memoria	Pila	Montón
Herencia	No compatible	Soportado
Mutabilidad	Inmutable por defecto	Mutable
Comportamiento de copia	Copia el valor	Copia la referencia
Rendimiento	Rápido	Más lento debido a ARC

Ejemplo:

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

print(person1.name) // "Alice" (inalterado)
print(person2.name) // "Bob" (modificado)
```

4. **¿Qué son las propiedades calculadas y en qué se diferencian de las propiedades almacenadas?**

- Propiedad almacenada: almacena un valor.
- Propiedad calculada: calcula dinámicamente un valor.

Ejemplo:

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

	// Propiedad calculada
	var area: Double {
		return width * height
	}
}

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

5. **¿Qué es la inferencia de tipos en Swift?**

Swift determina automáticamente el tipo de datos.

Ejemplo:

```
let x = 10 // Swift infiere x como Int
let y = "Hola" // Swift infiere y como String
```

6. **¿Cuál es la diferencia entre funciones mutantes y no mutantes?**

- La mutación es necesaria cuando se modifican las propiedades dentro de una estructura.

Ejemplo:

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

- Las clases no requieren mutación, ya que son tipos de referencia.

7. **¿Qué son las extensiones en Swift y cómo funcionan?**

Las extensiones agregan funcionalidad a los tipos existentes.

Ejemplo:

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

8. **¿Cómo se define y usa un protocolo en Swift?**

Un protocolo define un modelo de métodos y propiedades.

```
protocol Vehicle {
	var speed: Int { get set }
	func accelerate()
}

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

9. **¿Qué es Codable y cómo funciona en el análisis de JSON?**

Codable ayuda a codificar y decodificar JSON.

Ejemplo:

```
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. **Explicar los niveles de control de acceso de Swift (open, public, internal, fileprivate, private).**

Modificador	Accesibilidad
open	Accesible en cualquier lugar, subclasificable fuera del módulo
public	Accesible en cualquier lugar pero no subclasificable fuera del módulo
internal	Por defecto, accesible dentro del mismo módulo
fileprivate	Accesible dentro del mismo archivo

private	Accesible dentro del mismo ámbito

Ejemplo:

```
clase MyClass {
        private var secret = "Oculto"
}
```

11. ¿Qué es la programación orientada a protocolos (POP) en Swift?

Swift fomenta los protocolos en lugar de la herencia.

Ejemplo:

```
protocol Drivable {
        func drive()
}

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

12. ¿Cuál es la diferencia entre delegación y notificación?

Característica	Delegación	Notificación
One-to-One	✅	❌
One-to-Many	❌	✅
Use	protocol	NotificationCenter

Ejemplo:

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

13. ¿Cuál es la diferencia entre la herencia de clases y la conformidad del protocolo?

- Una clase solo puede heredar una superclase, pero se ajusta a varios protocolos.
- Los protocolos definen el comportamiento sin implementación.

Ejemplo:

```
class Animal { } // Superclase
protocol Walkable { }
class Dog: Animal, Walkable {} // El perro hereda de Animal y cumple con Walkable
```

14. ¿Cómo maneja Swift la herencia múltiple?

Swift no admite la herencia múltiple. En su lugar, use protocolos.

Ejemplo:

```
protocolo Flyable { }
protocolo Swimmable { }
```

```
class Bird: Flyable, Swimmable{} // Admite ambos comportamientos
```

15. ¿Cuáles son los tipos asociados en los protocolos Swift?

Permite que los protocolos definan un tipo de marcador de posición.

Ejemplo:

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

16. ¿Qué es un genérico en Swift y por qué es útil?

Los genéricos permiten un código flexible y reutilizable.

Ejemplo:

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

        a = b
        b = temp
}
```

17. Explique los principios SOLID en Swift.

Los principios de SOLID ayudan a escribir código escalable y mantenible:

1. **S** - Principio de responsabilidad única: Una clase debe tener una sola razón para cambiar.
2. **O** - Principio de apertura-cierre: El código debe estar abierto para la extensión pero cerrado para la modificación.
3. **L** - Principio de sustitución de Liskov: Los subtipos deben ser reemplazables por sus tipos base.
4. **I** - Principio de segregación de interfaz: Una clase no debe implementar métodos innecesarios que no utilice.
5. **D** - Principio de inversión de dependencias: Los módulos de alto nivel no deben depender de los módulos de bajo nivel.

18. ¿Qué es el recuento automático de referencias (ARC) en Swift?

ARC administra automáticamente la memoria mediante el seguimiento de referencias seguras a objetos. Cuando no quedan referencias, el objeto se desasigna.

Ejemplo:

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

ARC libera memoria automáticamente cuando no se utilizan instancias.

19. ¿Cuál es la diferencia entre referencias strong, weak y unowned?

Tipo de referencia	Propiedad	¿Evita las desasignaciones?	Caso de uso
strong	Predeterminado	Sí	Mantiene el objeto en la memoria
weak	Sin propiedad	No	Evita los ciclos de retención (por ejemplo, delegados)
unowned	Sin propiedad	No	Se usa cuando la referencia nunca debe ser nula

Ejemplo:

```
class Person {
	var pet: Pet?
}

class Pet {
	weak var owner: Person? // Evita la retención del ciclo
}
```

20. ¿Qué es un ciclo de retención y cómo se previene?

Un ciclo de retención ocurre cuando dos objetos tienen referencias fuertes entre sí, lo que evita la desasignación.

Solución: Utiliza referencias weak o unowned.

Ejemplo de ciclo de retención:

```
class A {
	var b: B?
}
class B {
	var a: A? // Ciclo de retención
}
```

Corrección con referencia weak:

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

21. ¿Qué es Grand Central Dispatch (GCD) y cómo se usa?

GCD es la API de Apple para la ejecución simultánea.

Ejemplo:

```
DispatchQueue.global(qos: .background).async {
	print("Tarea en segundo plano")
	DispatchQueue.main.async {
		print("Volver al hilo principal")
	}
}
```

22. ¿Qué es una cola de operaciones en iOS?

OperationQueue es una abstracción de nivel superior sobre GCD para una mejor administración de dependencias.

Ejemplo:

```
let queue = OperationQueue()

queue.addOperation {
        print("Operación ejecutada")
}
```

23. ¿Qué son async/await en Swift y cómo mejoran el manejo de la simultaneidad?

async/await simplifica la ejecución de código asincrónico.

Ejemplo:

```
func fetchData() async -> String {
        return "Data Received"
}
Task {
        let result = await fetchData()
        print(result)
}
```

24. ¿Qué es DispatchGroup y cuándo lo usa?

DispatchGroup se utiliza para sincronizar varias tareas.

Ejemplo:

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

25. ¿Qué es el Auto Layout y por qué es importante?

El Auto Layout es un sistema de diseño basado en restricciones que hace que la interfaz de usuario se adapte a diferentes tamaños de pantalla.

Ejemplo:

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

26. ¿Cuál es la diferencia entre frame y bounds?

Propiedad	Definición

frame	Posición y tamaño en relación con la superview
bounds	Posición y tamaño en relación consigo mismo

Ejemplo:

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

27. ¿Cuál es la diferencia entre XIB, Storyboard y la interfaz de usuario programática?

Método de interfaz de usuario	Pros	Contras
Storyboard	Interfaz de usuario visual de arrastrar y soltar	Difícil de gestionar en grandes proyectos
XIB	Componentes de interfaz de usuario reutilizables	No es ideal para aplicaciones completas
Interfaz de usuario programática	Control total	Más código para escribir

Ejemplo de interfaz de usuario programática:

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

28. ¿Qué es el ciclo de vida de UIViewController y cuándo se llama a los métodos?

Método del ciclo de vida	Cuando se ejecuta
viewDidLoad	Una vez, cuando se crea la vista
viewWillAppear	Antes de que aparezca la vista
viewDidAppear	Después de que aparezca la vista
viewWillDisappear	Antes de que la vista desaparezca
viewDidDisappear	Después de que la vista desaparezca

Ejemplo:

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

29. ¿Cuál es la diferencia entre UITableView y UICollectionView?

Componente	Caso de uso
UITableView	Lista de una sola columna
UICollectionView	Diseños basados en cuadrícula

Ejemplo:

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

30. ¿Cómo se pasan datos entre controladores de vista?

1. **Uso de Segue (prepareForSegue)**

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

2. **Uso de la delegación**

```
protocol DataPassingDelegate {
        func sendData(_data: String)
}
```

3. **Uso de cierres (Closures)**

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

4. **Uso de NotificationCenter**

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

31. ¿Qué es una fuente de datos difusible y cómo mejora las vistas de tabla?
Las fuentes de datos divisibles optimizan las actualizaciones de la interfaz de usuario mediante la administración automática de los cambios.

Ejemplo:

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

32. ¿Qué es URLSession y cómo se usa para realizar solicitudes de red?
URLSession es la API principal para realizar llamadas de red en Swift.

Ejemplo de una solicitud GET:

```
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 ?? " Error desconocido")
                return
        }
        print("Datos recibidos:", String(data: data, encoding: .utf8)!)
}
task.resume()
```

33. ¿Cómo se maneja el análisis de JSON en Swift?
Utiliza Codable para codificar/decodificar fácilmente los datos JSON.

Ejemplo:

```
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. ¿Cuál es la diferencia entre URLSession y las bibliotecas de redes de terceros como Alamofire?

Característica	URLSession	Alamofire
Complejidad	Más configuración manual	Más fácil, menos repetitivo
Dependencias	Sin dependencia externa	Requiere Alamofire
Flexibilidad	Control total	Abstracción de alto nivel

Ejemplo usando Alamofire:

```
importar Alamofire

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

35. ¿Qué es Combine y cómo mejora el manejo de datos?

Combine es el marco reactivo de Apple para manejar eventos asíncronos como respuestas API.

Ejemplo de uso de Combine para solicitudes de API:

```
import Combinar

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. ¿Qué es UserDefaults y cuándo debe usarlo?

UserDefaults se utiliza para almacenar pequeños pares clave-valor (por ejemplo, preferencias del usuario).

Ejemplo:

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

🚨 No se recomienda para datos grandes o confidenciales.

37. ¿Qué son los datos básicos y cómo funcionan?

Core Data es el marco de Apple para administrar gráficos de objetos y datos persistentes.

Ejemplo de guardado de datos en 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. ¿Cuál es la diferencia entre Core Data y Realm?

Característica	Core Data	Realm
Rendimiento	Más despacio	Rápido
Complejidad de la configuración	Se requiere más configuración	Configuración más sencilla
Sincronización	Sin sincronización incorporada	Sincronización incorporada

Ejemplo de guardar datos en Realm:

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

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

39. ¿Qué es la arquitectura MVC?

Componente	Responsabilidad
Modelo	Lógica empresarial, almacenamiento de datos
Vista	Interfaz de usuario (por ejemplo, etiquetas, botones)
Controlador	Controla la entrada del usuario y actualiza la vista

Ejemplo:

```
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. ¿Qué es la arquitectura MVVM? ¿Cómo mejora MVC?

MVVM presenta un ViewModel, que separa la lógica de la interfaz de usuario de la lógica de negocios.

Componente	Responsabilidad
Model	Datos y lógica empresarial
ViewModel	Procesa datos para la vista
View	Muestra la interfaz de usuario

Ejemplo:

```
class UserViewModel {
        var username: String

        init(user: User) {
                self.username = user.name.uppercased()
        }
}
```

41. ¿Qué es el patrón Singleton?

Un singleton garantiza que solo exista una instancia de una clase.

Ejemplo:

```
clase APIService {
        static let shared = APIService()
        private init() {} // Evita la creación de instancias
}
```

42. ¿Cómo se depuran las fugas de memoria en una aplicación iOS?

- Usar Instruments > Leaks tool
- Comprobar los ciclos de retención mediante Xcode Memory Graph

43. ¿Qué son los breakpoints y cómo se utilizan?

Los breakpoints pausan la ejecución para la depuración.

- Agregar un punto de interrupción simbólico (por ejemplo, en viewDidLoad)
- Usar comandos LLDB: po self // Imprimir objeto

44. ¿Cómo se almacenan de forma segura los datos confidenciales de los usuarios?

Usa Keychain, no UserDefaults.

Ejemplo:

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

45. ¿Cómo se mejora el tiempo de inicio de la aplicación?

- Minimice las operaciones pesadas en viewDidLoad
- Uso de subprocesos en segundo plano(DispatchQueue.global)
- Optimización de imágenes mediante recursos de imagen

46. ¿Cómo se manejan los Deep link en iOS?

Los Deep link permiten abrir páginas específicas de aplicaciones a través de URL.

Ejemplo:

```
func application(_ app: UIApplication, open url: URL) -> Bool {
        print("Abierto a través de \(url)")
        return true
}
```

47. ¿Qué es App Transport Security (ATS)?

ATS obliga a las aplicaciones a usar conexiones HTTPS por seguridad.

Para desactivar ATS (no recomendado):

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

48. ¿Qué son las pruebas unitarias y las pruebas de interfaz de usuario en iOS?

Tipo de prueba	Propósito
Prueba unitaria (Unit Test)	Pruebas de métodos individuales
Prueba de interfaz de usuario (UI Test)	Simula las interacciones del usuario

Ejemplo de una prueba unitaria:

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

49. ¿Cómo se distribuye una aplicación iOS?

1. App Store (a través de App Store Connect)
2. TestFlight (para pruebas beta)
3. Distribución empresarial (para aplicaciones internas)

50. ¿Cómo se manejan los bloqueos de aplicaciones en producción?

- Usar Crashlytics para registrar errores
- Implementación de NSExceptionHandler
- Recopilar informes de fallas de usuarios