iOS Interview Questions

By: Mahmoud Allam

1- What is `Completion Handler` ?

A `Completion handler` in Swift is a function that calls back when a task completes. This is why it is also called a callback function.

A callback function is passed as an argument into another function. When this function completes running a task, it executes the callback function.

2- What is `Typealias` ?

A `type alias` allows you to provide a new name for an existing data type into your program. After a type alias is declared, the aliased name can be used instead of the existing type throughout the program.

3- What is `defer` keyword ?

Swift's `defer` keyword lets us set up some work to be performed when the current scope exits

4- What is the difference between `Any` , `AnyObject` ?

- `Any` can represent an instance of any type at all, including function types , for both reference and value type .
- `AnyObject` can represent an instance of any class type, only for reference type .

5- What is `subscript` method ?

A **`subscript`** defines a shortcut to elements of a collection, list, or sequence. It can be defined in classes, structures, and enumerations to allow quick access to elements from a certain type.

6- What is `Member wise` initialization in Struct ?

A **`memberwise`** initializer is an initializer that is automatically generated by the compiler for structs that don't define a custom initializer in their declaration .

7- What is the difference between `Frame` and `Bounds` ?

The **`Bounds`** of an UIView is the rectangle, expressed as a location (x , y) and size (width , height) relative to its own coordinate system (0 , 0).

The **`Frame`** of an UIView is the rectangle, expressed as a location (x , y) and size (width , height) relative to the superview it is contained within.

8- What is the difference between `Upcast` and `Downcast` ?

The key difference between `Upcast` and `Downcast` in iOS Swift is that upcasting from a derived to a base class can be verified at compile-time and will never fail to compile and Downcasts, on the other hand, can fail to compile since the precise class is not always known. It is possible that the UIView you have is a UITableView or a UIButton.

9- What is the difference between `==` and `===` ?

`==` operator checks if their instance values are equal, **"equal to"**

`===` operator checks if the references point the same instance, **"Identical to" , (Same Reference)**

10- What does the Swift `mutating` keyword mean ?

structs are immutable. Meaning other variables can not change the values for instance of structure at any given point.

The `mutating` keyword is required for changing the values of self variables inside structure's function ONLY .

11- How to break out of multiple loop levels ?

By adding a label to the outer loop we can break out of both loops at once, like this :-

```
outerLoop: for number1 in numbers {
    for number2 in numbers {
        if number1 == number2 && number1 * number2 == 144 {
            print("Square found: \(number1)")
            break outerLoop
        }
    }
}
```

12- What is trailing closure syntax??

```
func travel(action: () -> Void) {
    print("I'm getting ready to go.")
    action()
    print("I arrived!")
}
```

By adding a label to the outer loop we can break out of both loops at once, like this :-

```
travel() {
    print("I'm driving in my car”)
```

13- What is `inout` parameters ?

All parameters passed into a Swift function are constants, so you can’t change them. If you want, you can pass in one or more parameters as `inout`, which means they can be changed inside your function, and those changes reflect in the original value outside the function.

14- What is difference between `as?`, `as!` and `as` in Swift ?

`as` can only be used for upcasting .

The conditional cast operator `as?` tries to perform a conversion, but returns `nil` if it can't. its result is optional.

The `as!` operator is for forced type conversion.

15- What's the difference between `Self` vs `self` ?

`Self` - use in protocol and extension declarations to refer to the eventual type that will conform to the protocol.

`self` (Lower Case) - explicit reference to the current type or instance of the type in which it occurs

16- What's the difference between a *protocol* and a *class* in Swift?

`class` defines what an object is.

`protocol` defines a behavior the object has.

17- How is an " `Unowned` reference" different from a "`Weak` Reference"??

`Weak` Reference:

A `Weak` reference is a reference that does not keep a strong hold on the instance it refers to, and so does not stop ARC from disposing of the referenced instance. Because weak references are allowed to have "no value", you must declare every weak reference as having an optional type. **(From Apple Docs)**

`Unowned` Reference:
Like weak references, an `Unowned` reference does not keep a strong hold on the instance it refers to. Unlike a weak reference, however, an unowned reference is assumed to always have a value. Because of this, an unowned reference is always defined as a non-optional type. **(From Apple Docs)**

When to Use Each:
Use a `Weak` reference whenever it is valid for that reference to become nil at some point during its lifetime. Conversely, use an `Unowned` reference when you know that the reference will never be nil once it has been set during initialization. **(From Apple Docs)**

18- When to use a set rather than an array in Swift?

`Sets` are especially useful when you need to ensure that an item only appears **once** in the set. **but not in order.**

`Array` in order but can be **duplicate items .**

19- What is the difference between `fileprivate` and `private`?

`file private` it can be read anywhere in the same file it was declared – even outside the type.

`private` property can only be read inside the type that declared it, or inside extensions to that type that were created in the same file.

20- Is there a way to create an *abstract class* in Swift?

by Protocol and Protocol extensions you can achieve the same behaviour .

First, you write a protocol that acts as an interface for all the methods that have to be implemented in all types that conform to it.
Then you can add default behaviour to all types that conform to it
You can now create new types by implementing the protocol.

21-What is Copy on Write (Cow) in Swift?

`Copy on write` is a common computing technique that helps boost performance when copying structures. To give you an example, imagine an array with 1000 things inside it: if you copied that array into another variable, Swift would have to copy all 1000 elements even if the two arrays ended up being the same.

This problem is solved using `Copy on write` : when you point two variables at the same array they both point to the same underlying data. Swift promises that structs like arrays and dictionaries are copied as values, like numbers, so having two variables point to the same data might seem to contradict that. The solution is simple but clever: if you modify the second variable, Swift takes a full copy at that point so that only the second variable is modified - the first isn't changed.

22-What is the difference between `open` and `public` keywords in Swift?

An `Open Class` is accessible and subclassable outside of the defining module.

An `Open Class` member is accessible and overridable outside of the defining module.

An `Public Class` is accessible but not subclassable outside of the defining module.

An `Public Class` member is accessible but not overridable outside of the defining module.

23-What is the use of `Hashable` protocol?

If an object conforms to the `Hashable` protocol, it needs to have A Hash Value , The Hash Value can be used to compare objects / uniquely identify the object.

You can compare objects in two ways:

1. `===` function. This checks object references (can only be used with classes). It checks if the left object has the same reference to the right object. Even if both objects have exactly the same property values BUT they do have a different reference, it returns false.

2. `==` function (Equatable protocol). It checks if the objects are equal to each other based on the static func ==. You can return the hashValue of the object. In that way, you can say objects are equal to each toher based on the properties, rather than reference.

24-What's the Fallible initializer `init?()` ?

the `init?()` usage is when the data that in initializer has been provided is insufficient or incorrect, and creation can't proceed. so it maybe come with nil value depending on a simple logic .

25-What is a `Variadic` function ?

the `Variadic` functions are functions that accept any number of parameters. The most common one in Swift is `print()` – most people use it to print a single value, but you can actually pass as many as you want . To make a variadic function of your own, just add `...` after any parameter

26-What’s the Difference Between Structures and Classes ?

Well in Swift There are two types of objects

1. `Struct`
2. `Class`

Main difference between them is

- `Struct` is **value** type : - When you make a copy of a value type, it copies all the data from the thing you are copying into the new variable. They are 2 separate things and changing one does not affect the other

- `Struct` has free initializer for you , you don't have to declare initializer if you do free initializer will be overwritten by your declared initializer .
- `Struct` Don't have deinitializer .
- `Struct` Cannot inherit from other struct.

- `Class` is **reference** type :- When you make a copy of a reference type, the new variable refers to the same memory location as the thing you are copying. This means that changing one will change the other since they both refer to the same memory location. The sample code below could be taken as reference.

- `Class` Must declare initializer (constructer)
- `Class` Has deinitializer
- `Class` Can inherit from other classes

(Note: - There’s more difference between struct and classes like struct stored in stack , classes instances stored in heap you can search about it)

27-What is Optional Chaining?

Swift provides us with a shortcut when using optionals: if you want to access something like `a.b.c` and `b` is optional, you can write a question mark after it to enable optional chaining: `a.b?.c`.

When that code is run, Swift will check whether `b` has a value, and if it's `nil` the rest of the line will be ignored – Swift will return `nil` immediately. But if it has a value, it will be unwrapped and execution will continue

28-What Is Optional Binding?

We use optional binding (using if let) to check if an optional contains the value. If the value exist , then we bind that value to the temporary local constant that only exist inside the if statement

29-What's the Difference Between `nil` and `None`?

There is no difference, as .none (basically it's Optional.none) and nil are equivalent to each other.

.none is an enum representation of the nil (absence of) value implemented by the Optional<T> enum.

30-How Can You Disallow a Class from Being Inherited?

You can prevent a class , method, property, or subscript from being overridden by marking it as `final`. Do this by writing the `final` modifier before the class , method, property, or subscript’s .

31-What Are `Lazy` Variables? When Should You Use One?

A `Lazy` **stored property** is calculated only when it is accessed for the first time ,

There are a few advantages in having a lazy property instead of a stored property.

1. The closure associated to the lazy property is executed only if you read that property. So if for some reason that property is not used (maybe because of some decision of the user) you avoid unnecessary allocation and computation.
2. You can populate a lazy property with the value of a stored property.
3. You can use self inside the closure of a lazy property

It is `var` and not `let` because, the value is not initialized during the initialization process. It is calculated later on. That's why a lazy stored property need to be a Variable and not a Constant.

32-What is silent Notification?

A `Silent` push notification reaches device, user does not know anything about the notification but his app gets the notification and app will be given some time to download new content and present it to the user, regardless to the state of the app (i.e. running or not running) .

33-What is the difference between `try`, `try?`, and `try!` in Swift?

A `try` catches the error and lets you handle by type of error .

A `try?` turns the result into an `Optional` that's `nil` if an error was thrown, you lose the type of error but know there was one .
A `try!` ignores the error completely and will just crash if one is thrown

34-What is the difference between an `escaping` closure and a `none`-`escaping` closure ?

A nonescape is A closure that's called within the function it was passed into, i.e. before it returns .

An escaping closure is a closure that's called after the function it was passed to returns. In other words, it outlives the function it was passed to.

35-What are opaque return type ?

Opaque types allow you to describe the expected return type without defining a concrete type. A common place where we use opaque types today is inside the body of a SwiftUI view:

```
var body: some View { ... }
```

36-What are `generics` and why are they useful ?

Swift provides '`generic`' features to write flexible and reusable functions and types. Generics are used to avoid duplication and to provide abstraction. Swift standard libraries are built with generics code. Swift 'Arrays' and 'Dictionary' types belong to generic collections. With the help of arrays and dictionaries the arrays are defined to hold 'Int' values and 'String' values or any other types.

37-What does the `CaseIterable` protocol do ?

`CaseIterable` is a protocol that automatically generates an array property of all cases in an enum. To enable it, all you need to do is make your enum conform to the **`CaseIterable`** protocol and at compile time Swift will automatically generate an **`allCases`** property that is an array of all your enum's cases, in the order you defined them.

38-What's the difference between `Raw values` and `Associated values` ?

If you want a case of an enum to stand for one simple constant literal value, use a `Raw value`.

```
enum PepBoy : String {
    case Manny = "Manny"
    case Moe = "Moe"
    case Jack = "Jack"
}
```

If you want a case of an enum to *carry* arbitrary value(s) of a predefined type, use an `Associated value`.

```
enum Error : ErrorType {
    case Number(Int)
    case Message(String)
}
```

37-What is Operator Overloading?

Swift supports operator overloading, which is a fancy way of saying that what an operator does depends on the values you use it with. For example, + sums integers like this:

```
let meaningOfLife = 42
let doubleMeaning = 42 + 42
```

But + also joins strings, like this:

```
let fakers = "Fakers gonna "
let action = fakers + "fake"
```

Remember, Swift is a type-safe language, which means it won't let you mix types. For example, you can't add an integer to a string because it doesn't make any sense.

38-How to define optional methods in swift protocols?

We can make an optional methods in protocols by Two ways :-

1 - Using default implementations (preferred).

2- Using @objc optional.

39-How background iOS app gets resumed in foreground?

When user launches an app that is currently in background, the system moves app to the inactive state and then to the active state.

40-What are the steps involved when app enter to foreground after device rebooted?

When user launches an app for the first time or after device reboot or after system terminate the app, the system moves app to the active state.

41-Which is the app state when device is rebooted?

Not Running state.

42-When app is running but not receiving event. In which state app is in?

Inactive state.

43-How an iOS app responds to interrupts like SMS, Incoming Call, Calendar, etc.?

Application moves to the inactive state temporarily and it remains in this state until the user decides whether to accept or ignore the interruption.

If the user ignores the interruption, the application is reactivated.

If the user accepts the interruption, the application moves into the suspended state.

44-Explain ViewController Life cycle?

1- ViewDidLoad - Called when you create the class and load from xib. Great for initial setup and one-time-only work.

2- ViewWillAppear - Called right before your view appears, good for hiding/showing fields or any operations that you want to happen every time before the view is visible. Because you might be going back and forth between views, this will be called every time your view is about to appear on the screen.
3- ViewDidAppear - Called after the view appears - great place to start an animations or the loading of external data from an API.

4- ViewWillDisappear/DidDisappear - Same idea as ViewWillAppear/ViewDidAppear.

5- ViewDidUnload/ViewDidDispose - In Objective-C, this is where you do your clean-up and release of stuff, but this is handled automatically so not much you really need to do here.

45-Content Hugging priority vs Content Resistance priority?

Hugging => content does not want to grow

Compression Resistance => content does not want to shrink

46-What is the difference between protocol and delegate?

Protocol is a set of methods (either optional or required) that would be implemented by the class which conforms to that protocol. While, delegate is the reference to that class which conforms to that protocol and will adhere to implement methods defined in protocol.

47-is it possible that an iOS Application can have more than one window?

Yes it can have multiple windows. Just only one to be displayed at a time .

48-Protocol & Delegation vs Notification Center?

You can think of delegates like a telephone call. You call up your buddy and specifically want to talk to them. You can say something, and they can respond. You can talk until you hang up the phone. Delegates, in much the same way, create a link between two objects, and you don't need to know what type the delegate will be, it simply has to implement the protocol , <ONE TO ONE Communication> .

On the other hand, NSNotifications are like a radio station. They broadcast their message to whoever is willing to listen. The radio station can't receive feedback from it's listeners (unless it has a telephone, or delegate). The listeners can ignore the message, or they can do something with it. NSNotifications allow you to send a message to any objects, but you won't have a link between them to communicate back and forth. If you need this communication, you should probably implement a delegate. Otherwise, NSNotifications are simpler and easier to use, but may get you into trouble. <ONE TO MANY Communication>

49-What is Automatic Reference Counting (ARC)?

it’s Apple way of handling memory of objects for developers , it’s keeping count for each object of how many strong references are pointing to that object . the object removed from memory when the count equal zero.

50-When is Memory Leak happening , how to fix it and how to find?

it’s Happening when two object got strong references from each others , so that they can’t be removed from memory because arc keep tracking the strong references of each object and removing them when the count equal zero .

it can be fixed by making one of those objects contains a weak ref instead of strong reference.

we can detect memory leaks from the visual memory debugger or from profiling memory leaks.

51-What is GCD?

GCD is a library that provides a low level and object based API to run tasks concurrently while managing threads behind the scenes. GCD abstracts away thread management code and moves it down to the system level, exposing a light API to define tasks and execute them on an appropriate dispatch queue.

52-What is dispatch queue?

A dispatch queue is responsible for executing a task in the FIFO order.

53-Explain main thread and it’s usages?

In iOS, application main thread is a globally available serial queue. Main thread should be used for all views and interface elements must not be blocked by long running tasks. Developer should avoid functions using main thread to load data, images, etc.

54-What is serial queue?

The serial queue can run tasks one at a time and it needs to wait for the started tasks to finish.

55-What is concurrent queue?

The concurrent queue can run as many tasks as it can without waiting for the started tasks to finish.

56-Explain three GCD queues?

GCD provides three type of queues –

1- Main queue: Serial queue – It runs on the main thread.

2- Global queue: Concurrent queue – It runs with different priorities and shared by the intire system.

3- Custom queue: Serial/ Concurrent queue.

57-What is GCD’s Quality of Service?

Quality of Service is the priority to perform task in GCD. If the task has higher quality of service than other it will be handled before lower quality of service.

Quality of Services are ranked from highest to lowest –

1- User Interactive: Work that happens on the main thread, such as animations or drawing operations.

2- User Initiated: Work that the user kicks off and should yield immediate results. This work must be completed for the user to continue.

3- Utility: Work that may take a bit and doesn’t need to finish right away. Analogous to progress bars and importing data.

4-Background: This work isn’t visible to the user. Backups, syncs, indexing, etc.

58-What is race condition?

A race condition occurs when two or more threads can access shared data and they try to change it at the same time.

59-What is priority inversion?

A Priority inversion is a critical condition in threading where a low priority thread blocks a high priority thread from executing and make the assigned priorities meaningless for the thread.

60-What is deadlock?

A deadlock occurs when two or sometimes more tasks wait for the other to finish, and neither ever does.

61-What is concurrency?

Concurrency is the structural way to run multiple tasks at the same time.

62-What is NSOperation?

NSOperation is built on top of GCD. While using NSOperation developer can add dependency among various operations and re-use, cancel and suspend operations. Of course the same thing can also be archived through GCD but it will add extra overhead.

63-What is semaphore?

Semaphores gives us the ability to control access to a shared resource by multiple threads. A semaphore consist of a threads queue and a counter value. Counter value is used by the semaphore to decide if a thread should get access to a shared resource or not. The counter value changes when we call signal() or wait() functions.

64-What is design pattern and explain briefly some commonly used Cocoa design patterns?

Design patterns are reusable solutions to common problems in software design. Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Commonly used Cocoa design patterns are –

Creational Design Pattern: Singleton, Factory

Structural Design Pattern: Decorator, Adapter, Facade

Behavioral Design Pattern: Observer, Memento.

65-What is Singleton Pattern and why should we avoid over using it?

The singleton pattern(Creational) guarantees that only one instance of a class is instantiated for the lifetime of the application. It is very easy pattern to implement but it also has bad reputation to misuse. Singleton kills the idea of "separation of concern" as it can be accessed from everywhere(They are generally used as a global instance). Singleton violates the single responsibility principle .

66-What is Singleton Pattern and why should we avoid over using it?

Singletons are a problem from a testing perspective. They tend to make isolated unit-tests difficult to write because one test can change the global value which is not valid for other tests.

67-What is Decorator Design Pattern ?

The decorator design pattern(Structural) dynamically adds behaviours and responsibilities to an object without modifying its code. Unlike inheritance, decorated objects are not limited by their parent classes. Putting in other terms, a client has control over how and when to decorate the component. This pattern is simply achieved by using protocol in Swift.

68-What is factory method in Swift?

Factory method is a creational design pattern which solves the problem of creating objects without specifying their concrete classes.

69-What is Facade Design Pattern ?

The facade design pattern provides a single interface to a complex subsystem. Instead of exposing the user to a set of classes and APIs, you only expose one simple unified API. In Swift, you can achieve this pattern using extensions and delegation.

67-What are the basic differences between creational, structural and behavioural pattern?

The creational pattern provides way to create objects while hiding the creational logic which offers the flexibility for creating objects based on different use cases.

The structural pattern helps us to manage classes and object together to create larger components.

The behavioural pattern helps us to provide better communication between objects and increase flexibility between object.

71-What is Adapter Pattern?

The adapter design pattern allows two objects, with related functionalities, to work together, even when they have incompatible interfaces. Adapter allows the objects to cooperate with other objects where they could not normally work with due to different interfaces. It is a structural design pattern which is useful for composing classes and objects into a larger system. Swift does not support multiple inheritance but Swift supports conformance to multiple protocols, you can implement adapter pattern by using protocols.

72-What is Observer Pattern ?

Observer pattern is a behavioural design pattern where the objects can notify other objects about the changes in their state. The Observer pattern provides a way to subscribe and unsubscribe to and from these events for any object that implements a subscriber interface.

We can implements the observer pattern in two ways in Swift – Notifications and Key-Value Observing.

73-What is the delegation pattern?

Delegation is a design pattern that enables a class or structure to hand off (or delegate) some of its responsibilities to an instance of another type. The delegating object typically keeps a reference to the other object (delegate) and sends a message to it at the appropriate time.

74-What is POP?

Protocol-Oriented Programming is a new programming paradigm ushered in by Swift 2.0. In the Protocol-Oriented approach, we start designing our system by defining protocols. We rely on new concepts: protocol extensions, protocol inheritance .

In Swift, value types are preferred over classes. However, object-oriented concepts don’t work well with structs and enums: a struct cannot inherit from another struct, neither can an enum inherit from another enum.

On the other hand, value types can inherit from protocols, even multiple protocols. Thus, with POP, value types have become first-class citizens in Swift.

75-How would you securely store private user data offline on a device? What other security best practices should be taken?

If the data is extremely sensitive then it should never be stored offline on the device because all devices are crackable.

The keychain is one option for storing data securely. However it's encryption is based on the pin code of the device. User's are not forced to set a pin, so in some situations the data may not even be encrypted. In addition the users pin code may be easily hacked.

76-Could you explain what is the difference between Delegate and KVO?

Both are ways to have relationships between objects. Delegation is a one to one relationship where one object implements a delegate protocol and another uses it and sends messages to assuming that those methods are implemented since the receiver promised to comply to the protocol. KVO is many-to-many relationship where one object could broadcast a message and one or multiple other objects can listen to it and react.