*1.)What is the difference between ANY and ANYOBJECT ?*

According to Apple’s Swift documentation:

* **Any** can represent an instance of any type at all, including function types and optional types.
* **AnyObject**can represent an instance of any class type.

***2.)How many different ways to pass data in Swift ?***

There are many different ways such as Delegate, KVO, Segue, NSNotification, Target-Action and Callbacks.

***3.)What is the difference between Delegates and Callbacks ?***

The difference between delegates and callbacks is that with delegates, the NetworkService is telling the delegate “There is something changed.” With callbacks, the delegate is observing the NetworkService.

***4.)What’s the difference between optional nil and .none?***

There is no difference. **optional.none** (**.none** for short) is the correct way of initializing an optional variable lacking a value, whereas **nil** is just syntactic sugar(syntax formation/explanation related) for **.none**.

***5.)How to remove derived data in xcode ?***

First close Xcode 🡪 Open Terminal 🡪 paste

[rm -rf ~/Library/Developer/Xcode/DerivedData] without using square bracket 🡪 Open Xcode again 🡪 All the Derived data will Remove .

or,

Shift +Alt+Command + k

***6.)Where do we use Dependency Injection ?***

We use a storyboard or xib in our iOS app, then we created IBOutlets. IBOutlet is a property related to a view. These are injected into the view controller when it is instantiated, which is essentially a form of Dependency Injection.

There are some forms of dependency injection: constructor injection, property injection and method injection.

**7.)***IOS Characteristics*

|  |  |
| --- | --- |
| **Criteria** | **Result** |
| Type of Operating System | Apple Proprietary based on Macintosh OS X |
| OS Fragmentation | Tightly integrated with Apple devices |
| Security | Heightened(put up/increase) security guaranteed |

*8.)What is Completion Handler?*

Completion handlers are super convenient when our app is making an API call and we need to do something when that task is done, like updating the UI to show the data from the API call. We’ll see completion handlers in Apple’s APIs like dataTaskWithRequest and they can be pretty handy in our own code.  
  
The completion handler takes a chunk of code with 3 arguments:(NSData?, NSURLResponse?, NSError?) that returns nothing: Void. It’s a closure.  
  
The completion handlers have to marked @escaping since they are executed some point after the enclosing function has been executed.

*9.)What is Multithreading? Explain in brief.*

Multithreading allows the processor to create concurrent threads it can switch between, so multiple tasks can be executed at the same time. It appears as - if the two threads are executed at the same time, because the processor switches rapidly between executing them.

concurrent means - (at a time).

* 1. NSThred
  2. NSOperation & NSOperationQueue
  3. GCD F/w (FrameWork) [Dispatch\_queue]

**(i.)NSThread** :-

**NSThread** creates a new low-level thread which can be started by calling start method.

Syntax :

var myThread = Thread(target: self, selector: #selector(self.myThreadMainMethod), object: nil)

myThread**.**start()

(ii.)NSOperation :- **NSOperation** is an abstract class, designed for subclassing. Each subclass represents a specific **task .**

Syntax :

let operation = NSOperation(block: { () -> void in

})

operationQueue.addOperation(operation)

**(iii.)NSOperationQueue: NSOperationQueue** allows a pool of threads to be created and used to execute NSOperations in parallel. NSOperations can also be run on the main thread by asking NSOperationQueue for the mainQueue.

Syntax:

var myQueue = NSOperationQueue()

myQueue.addOperation(anOperation)

myQueue.addOperationWithBlock({() -> Void in

})

* 1. **GCD or Grand Central Dispatch[**Dispatch\_queue**] : GCD or Grand Central Dispatch**  is a modern feature of swift that provides a rich set of methods and API's to use in order to support common multi-threading tasks. GCD provides a way to queue tasks for dispatch on either the main thread, a concurrent queue (tasks are run in parallel) or a serial queue (tasks are run in FIFO order).

Syntax :

let queue = dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_HIGH, 0)

dispatch\_async(queue) {

}

#### Serial Queues(Main Queues/ **deterministic** Queues)

When we choose to create a queue as serial queue, the queue can only execute one task at a time. All tasks in the same serial queue will respect each other and execute serially. However, they don’t care about tasks in separate queues which means that we can still execute tasks concurrently by using multiple serial queues. For example, we can create two serial queues, each queue executes only one task at a time but up to two tasks could still execute concurrently.

The benefits of using serial queues are:

1.Guaranteed serialized access to a shared resource that avoids race condition.

2.Tasks are executed in a predictable order. When we submit tasks in a serial dispatch queue, they will be executed in the same order as they are inserted.

3.We can create any number of serial queues.

Concurrent Queues(Background Queues*/* **nondeterministic** Queues)

As the name suggests, concurrent queues allows we to execute multiple tasks in parallel. The tasks (blocks of codes) starts in the order in which they are added in the queue. But their execution - all occur concurrently and they don’t have to wait for each other to start. Concurrent queues guarantee that tasks start in same order but we will not know the order of execution, execution time or the number of tasks being executed at a given point.

For example, we submit three tasks (task #1, #2 and #3) to a concurrent queue. The tasks are executed concurrently and are started in the order in which they were added to the queue. However, the execution time and finish time vary. Even it may take some time for task #2 and task #3 to start, they both can complete before task #1. It’s up to the system to decide the execution of the tasks.

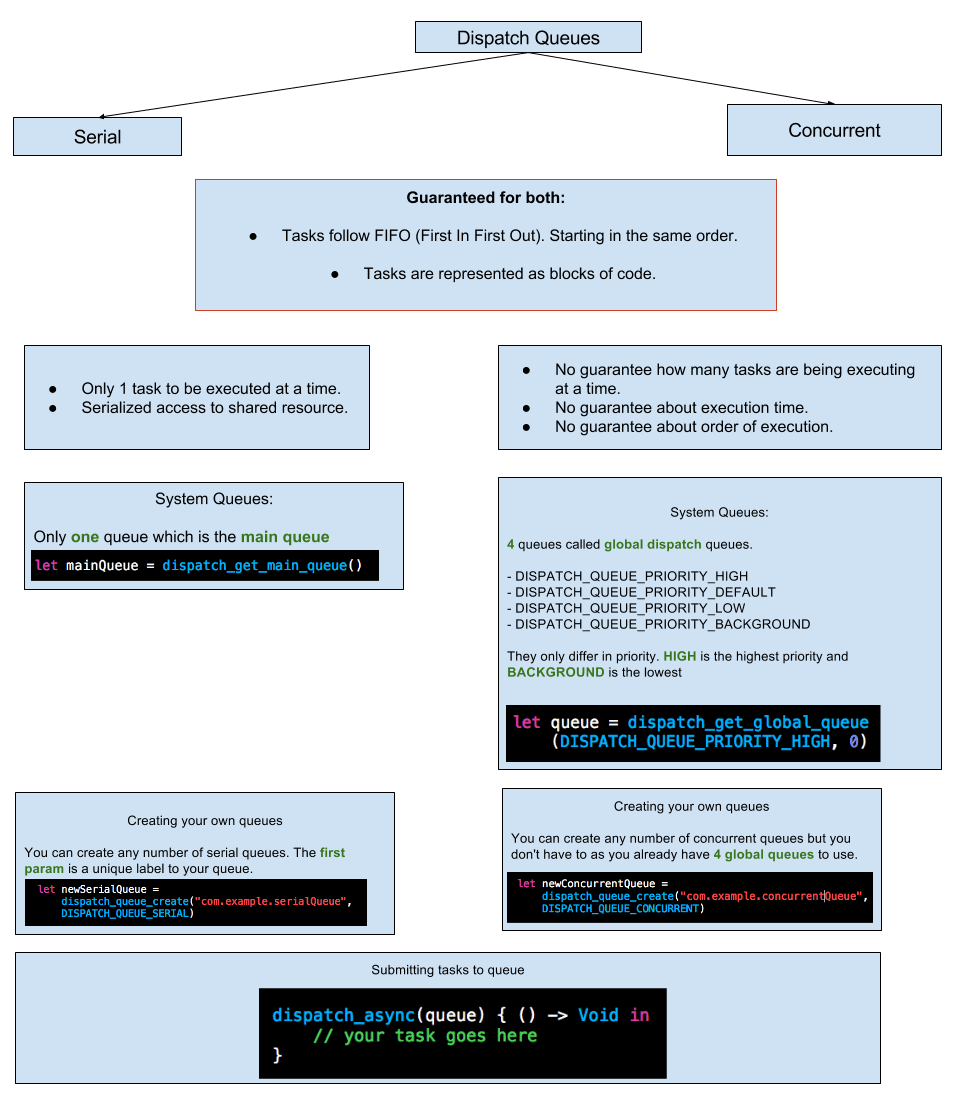
The benefits of using concurrent queues are:

(i.)Tasks are executed in random order.

(ii.) Faster compared to serial queues.

**Queue Opearions**

* **async** - **concurrent**: the code runs on a background thread. Control returns immediately to the main thread (and UI). The block can't assume that it's the only block running on that queue.
* **async** - **serial**: the code runs on a background thread. Control returns immediately to the main thread. The block can assume that it's the only block running on that queue.
* **sync** - **concurrent**: the code runs on a background thread but the main thread waits for it to finish, blocking any updates to the UI. The block can't assume that it's the only block running on that queue (we could have added another block using async a few seconds previously).
* **sync** - **serial**: the code runs on a background thread but the main thread waits for it to finish, blocking any updates to the UI. The block can assume that it's the only block running on that queue.



*10.) What is Concurrency ?* Concurrency is dividing up the execution paths of our program so that they are possibly running at the same time. The common terminology: process, thread, multithreading and others.

Terminology:

* **Process:** An instance of an executing app.
* **Thread:** Path of execution for code.
* **Multithreading:** Multiple threads or multiple paths of execution running at the same time.
* **Concurrency:** Execute multiple tasks at the same time in a scalable manner.
* **Queues:** Queues are lightweight data structures that manage objects in the order of First-in, First-out (FIFO).

*11.) Expain Grand Central Dispatch (GCD).* **GCD** is a library that provides a low-level and object-based API to run tasks concurrently while managing threads behind the scenes.

Terminology:

* **Dispatch Queue:** A dispatch queue is responsible for executing a task in the first-in, first-out order.
* **Serial Dispatch Queue:** A serial dispatch queue runs one task at a time.
* **Concurrent Dispatch Queue:** A concurrent dispatch queue runs as many tasks as it can without waiting for the started tasks to finish.
* **Main Dispatch Queue:** A globally available serial queue that executes tasks on the application’s main thread.

*12.) Explain Readers-Writers.* Multiple threads reading at the same time while there should be only one thread writing. The solution to the problem is a readers-writers lock which allows concurrent read-only access and an exclusive write access.

Terminology:

* **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.
* **Deadlock:** A deadlock occurs when two or sometimes more tasks wait for the other to finish and neither ever does.
* **Readers-Writers problem:** Multiple threads reading at the same time while there should be only one thread writing.
* **Readers-writer lock:** This lock allows concurrent read-only access to the shared resource while write operations require exclusive access.
* **Dispatch Barrier Block:**Dispatch barrier blocks create a serial-style bottleneck when working with concurrent queues.

*13.)Explain NSOperation — NSOperationQueue — NSBlockOperation.*

**NSOperation:**  Adds a little extra overhead compared to GCD, but we can add dependency among various operations and re-use, cancel or suspend them.

**NSOperationQueue:**It allows a pool of threads to be created and used to execute NSOperations in parallel. Operation queues aren’t part of GCD.

**NSBlockOperation:** Allows we to create an NSOperation from one or more closures. NSBlockOperations can have multiple blocks, that run concurrently.

*14.)Explain Main Thread Checker.*

The Main Thread Checker is a new tool launched with Xcode 9 which detects the invalid use of Apple’s frameworks like UIKit, AppKit etc that supposed to be used from main thread but accidentally used in the background thread. The effect of the invalid usage can result in missed UI updates, visual defects, data corruption and crashes.

*15.)Why do we use synchronized ?*

synchronized guarantees that only one thread can be executing that code in the block at any given time.

*16.)Explain KVC — KVO.*

**KVC**  stands for Key-Value Coding. It’s a mechanism by which an object’s properties can be accessed using string’s at runtime rather than having to statically know the property names at development time.

**KVO**stands for Key-Value Observing and allows a controller or class to observe changes to a property value. In KVO, an object can ask to be notified of any changes to a specific property, whenever that property changes value, the observer is automatically notified.

*17.)**What is Observer Pattern ?*

In the Observer pattern, one object notifies other objects of any state changes. Cocoa implements the observer pattern in two ways: **Notifications** and **Key-Value Observing (KVO)**.

*18.)* ***what is defer?***

The term defer can simply be interpreted as a keyword which provides a block of code that will be executed in the case when execution is leaving the current scope.

*19.)What is NotificationCenter ?*

NotificationCenter is an observer pattern. The **NSNotificationCenter** singleton allows us to broadcast information using an object called **NSNotification**.

The biggest difference between **KVO** and **NotificationCenter** is that KVO tracks specific changes to an object, while **NotificationCenter** is used to track generic events.

20.)*What is Protocol?*

A protocol defines a blueprint of methods, properties and other requirements that suit a particular task or piece of functionality. The protocol can be adopted by a class, structure or enumeration to provide an actual implementation of those requirements.

*21.)What is Sequence protocol ?*

A sequence is a list of values that we can step through one at a time. The most common way to iterate over the elements of a sequence is to use a [for-in](http://swiftdoc.org/v3.0/protocol/Sequence/) [loop](http://swiftdoc.org/v3.0/protocol/Sequence/) . Such type of protocol is called sequence protocol.

*22.)Explain IteratorProtocol.*

The iterator protocol is tightly linked with the sequence protocol. Sequences provide access to their elements by creating an iterator, which keeps track of its iteration process and returns one element at a time as it advances through the sequence.

23.)*Explain CodingKey*enum  *Protocol.*

The **CodingKey** enum **Protocol**lets we rename specific properties in case the serialized format doesn’t match the requirements of API. CodingKeys should have nested enum.

*24.)What’s the difference between MKAnnotation and MKPointAnnotation?*

MKAnnotation is a protocol. Typically, we will create a NSObject subclass that implements this protocol. Instances of this custom class will then serve as our map annotation.

MKPointAnnotation is a class that implements MKAnnotation. We can use it directly if we want our own business logic on the annotation.

*25.)Explain @objc inference . //Inherited from Objective - C for Swift*

We can tag a Swift declaration with ***@objc*** to indicate that it should be available to Objective-C. In Swift many declarations were automatically *inferred* to be made available to Objective-C. The most common place for this is any Swift method we want to refer to using a selector.

*26.)Explain @objcMembers inference. //Inherited from Objective - C for Swift*

When we declare this class as having @objcMembers everything in the class will be exposed to the Objective-C runtime. This is the same as writing implicitly @objc in front of the function.

*27.)Explain type inference .//Swift Concept*

Swift uses type inference to work out the appropriate type. Type inference enables a compiler *to deduce the type of a particular expression automatically when it compiles our code*, simply by examining the values we provide.

*28.)What is the purpose of the reuseIdentifier ?*

The purpose of reuseIdentifier  is - Reusability of an already allocated object.

29*.) What is Core Data ?*

Core Data is not an object-relational mapper(ORM) ,Nor a database. Instead, Core Data is an object graph manager which also has the ability to persist object graphs to a persistent store, on a disk.

*30.)What is a managed object context ?*

A managed object context represents a single object space, or scratch pad, in a Core Data application.

*31.)What is category and when it is used ?*

A category is a way of adding additional methods to a class without extending it. It is often used to add a collection of related methods. A common use case is to add additional methods to built-in classes in the Cocoa frameworks.

## *32.)What is the difference between atomic and nonatomic properties? Which is the default for synthesized properties?*

Properties specified as atomic always return a fully initialized object. This also happens to be the default state for synthesized properties. But, if we have a property for which we know that retrieving an uninitialized value is not a risk (e.g. if all access to the property is already synchronized via other means).Then setting it to nonatomic can give us better performance than atomic.

***33.)What is the difference between Non-Escaping and Escaping Closures ?***

**Escaping closure means**, inside the function, we can still run the closure (or not); the extra bit of the closure is stored some place that will outlive (long lasting) the function. There are several ways to have a closure escape its containing function:

**(i.)Asynchronous execution**: If we execute the closure asynchronously on a dispatch queue, the queue will hold onto the closure for us. We have no idea when the closure will be executed and there’s no guarantee that - it will complete before the function returns.

**(ii.)Storage**: Storing the closure to a global variable, property or any other bit of storage that lives on past the function call means the closure has also escaped.

The lifecycle of a non-escaping closure is simple:

1. Pass a closure into a function
2. The function runs the closure (or not)
3. The function returns

*34.)Which Function returns Different Data Types? Example: Tuple contaions String , Array and Dictionary together ?Receive this return value and display ?*

func definedByMe(name: String, age: Int, myArray: [Int], myDict: [String:String]) -> (String, Int, [Int],[String:String]) {

let msg = "Good Morning! " + name

var userage = age

if age > 30{

userage = 29

}

return(msg, userage,myArray,myDict)

}

let output: (String, Int,[Int],[String:String]) = definedByMe(name: "Manju", age: 80 ,myArray: [1,2,3],myDict: ["first key":"first value","second key":"second value"])

print(output.0)

print("Your age is \(output.1)")

print("Array values \(output.2)")

print("Dict values \(output.3)")

o/p(using playground) –

Good Morning! Manju

Your age is 29

Array values [1, 2, 3]

Dict values ["first key": "first value", "second key": "second value"]

(ii.) func definedByMe(name: String, age: Int, myArray: [Int], myDict: [String:String]) -> (first:String, Second:Int, third:[Int],fourth:[String:String]) {

let msg = "Good Morning! " + name

var userage = age

if age > 30{

userage = 29

}

return (msg, userage,myArray,myDict)

}

let output = definedByMe(name: "Manju", age: 80 ,myArray: [1,2,3],myDict: ["first key":"first value","second key":"second value"])

print(output.first)

print("Your age is \(output.Second)")

print("Array values \(output.third)")

print("Dict values \(output.fourth)")

o/p(using playground):

Good Morning! Manju

Your age is 29

Array values [1, 2, 3]

Dict values ["second key": "second value", "first key": "first value"]

*35.)Explain all the methods of ApplicationDelegate .*

(i.)func application(\_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:gap [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

// Override point for customization after application launch.

return true

}

(ii.) func applicationWillResignActive(\_ application: UIApplication) {

// Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.

// Use this method to pause ongoing tasks, disable timers and invalidate graphics rendering callbacks. Games should use this method to pause the game.

}

(iii.)func applicationDidEnterBackground(\_ application: UIApplication) {

// Use this method to release shared resources, save user data, invalidate timers and store enough application state information to restore our application to its current state ; in this case - it is terminated later.

// If our application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.

}

(iv.) func applicationWillEnterForeground(\_ application: UIApplication) {

// Called as part of the transition from the background to the active state; here we can undo many of the changes made on entering the background.

}

(v.)func applicationDidBecomeActive(\_ application: UIApplication) {

// Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.

}

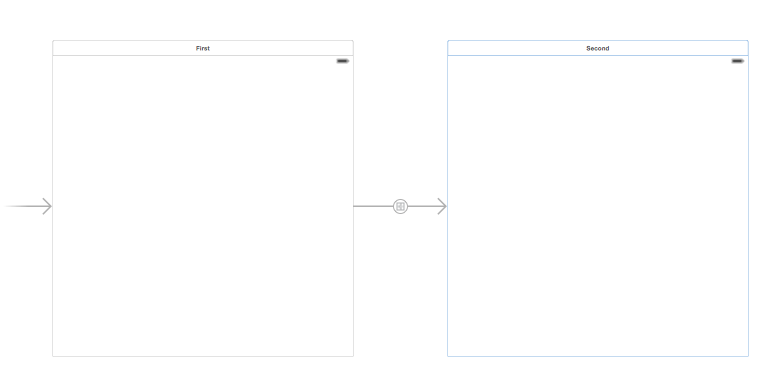
(vi.) func applicationWillTerminate(\_ application: UIApplication) {

// Called when the application is about to terminate. Save data if appropriate.

}

# *36.)Explain Storyboard, Scene and Segue.*

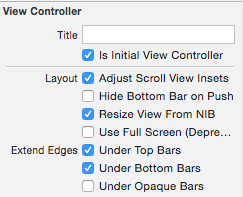
Storyboard in iOS help us to design user interface for our App. A storyboard can contain one or more Scenes (View Controllers) and the connection or relationship between two scenes are known as Segue. This is how a typical storyboard with Scenes and Segue looks.



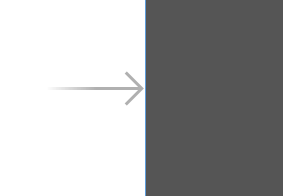
In the above example storyboard contains two scenes, first and second. These scenes are connected to each other by Show Segue.

### Initial ViewController

App needs to know the ViewController that will be displayed as the initial screen. We can set any controller as the initial scene using the option available as part of the **Attributes Inspector.**

[](http://rshankar.com/wp-content/uploads/2015/06/1435222038_full.png)

The storyboard will display an arrow in front of Initial View Controller as shown in below screenshot. We can also drag and drop this arrow to change the Initial View Controller.

[](http://rshankar.com/wp-content/uploads/2015/06/1435222086_full.png)

### Segues

A segue is a relationship set between two view controllers. This can be done by pressing Control then drag and drop from one View Controller to another. The connection is between an object in one scene that a user touches to initiate the transition and a scene or a storyboard reference that is the target of the transition.

On the storyboard, segues appear as an arrow between scenes. The arrow includes a symbol for the type of segue. The below screenshot shows a push segue.



### Types of Segues

Listed below are the different types of segues

* Show
* Show Detail
* Present Modally
* PopOver Presentation
* Custom

| **Symbol** | **Type** | **Description** |
| --- | --- | --- |
| https://help.apple.com/xcode/mac/8.0/en.lproj/Art/SB_segue_push.png | Show | Present the content in the detail or master area depending on the content of the screen.  If the app is displaying a master and detail view, the content is pushed onto the detail area. If the app is only displaying the master or the detail, the content is pushed on top of the current view controller stack. |
| https://help.apple.com/xcode/mac/8.0/en.lproj/Art/SB_segue_push.png | Show Detail | Present the content in the detail area.  If the app is displaying a master and detail view, the new content replaces the current detail. If the app is only displaying the master or the detail, the content replaces the top of the current view controller stack. |
| https://help.apple.com/xcode/mac/8.0/en.lproj/Art/SB_segue_modal.png | Present Modally | Present the content modally. |
| https://help.apple.com/xcode/mac/8.0/en.lproj/Art/SB_segue_popover.png | Present as Popover | Present the content as a popover anchored to an existing view. |
| https://help.apple.com/xcode/mac/8.0/en.lproj/Art/SB_segue_custom.png | Custom | Create our own behaviors by using a custom segue. |

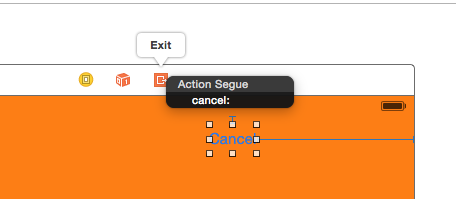
### 

### Unwind Segues

Only the Show segue has a back button in the navigation bar of the child controller which takes us back to the Parent controller. For rest of the segues, we can use unwind segues to dismiss the child controller. Unwind segue can be created by first adding a IBAction with parameter type as UIStoryboardSegue to the parent View Controller.

|  |
| --- |
| // Unwind Segue      @IBAction func cancel(segue:UIStoryboardSegue {      } |

Add a button to the child view controller press Control then drag and drop the button to the Exit button available at the top of the ViewController. Now select the Action Segue which was added to the parent view controller.

[](http://rshankar.com/wp-content/uploads/2015/06/1435222821_full.png)

One of the common question that gets asked is “**how to transfer data between two view controllers**“. Create a property in the child view controller and set the value for this property in the prepareForSegue function of the parent view controller.

|  |
| --- |
| // ViewController Transition        override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {          if segue.identifier == "detail" {              let childViewController = segue.destinationViewController as! SecondViewController              childViewController.studentName = nameTextField.text          }      } |

In the above prepareForSegue function, we check whether segue identifier is equal to detail or not. The identifier name can be provided under the Storyboard Segue in Attributes Inspector. Then we set the value for studentName property of the SecondViewController. By this way, we can transfer data between the View Controllers.

In the SecondViewController, we should be able to access this property value and perform the required action. The below viewDidLoad code snippet(fragment) - access the property value and displays it in a label.

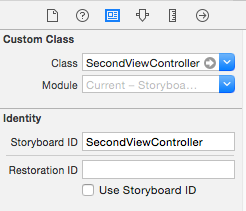
|  |
| --- |
| class SecondViewController: UIViewController {      var studentName:String?      @IBOutlet var displayName:UILabel! /\* Note : This coding line doesn’t written with weak var, it means  when we write weak var then memory can remove weak whenever they want but in case of only var , it’s a strong  var and to remove strong var we have to remove it manually from the memory . \*/   override func viewDidLoad() {          super.viewDidLoad()          // Do any additional setup after loading the view.          if let name = studentName {              displayName.text = name          }      }  } |

### Non Segue Transition

Another alternate way to transition between the View Controllers - is to use instantiateViewControllerWithIdentifier function of storyboard.

|  |
| --- |
| // Non Segue      @IBAction func nonSegueTransition() {          let childViewContoller = storyboard?.instantiateViewControllerWithIdentifier("SecondViewController") as!  SecondViewController          childViewContoller.studentName = nameTextField.text          presentViewController(childViewContoller, animated: true, completion: nil)      } |

The above code snippet(tukra) provides the same behaviour as prepareForSegue function. This function can be called on tap of a button and this requires an identifier to specified for the View Controller.

[](http://rshankar.com/wp-content/uploads/2015/06/1435227255_full.png)

*37.)What is the purpose of using IBAction and IBOutlet ?*

IBAction and IBOutlet are macros defined to denote variables and methods that can be referred to in Interface Builder.

IBAction resolves to void and IBOutlet resolves to nothing, but they signify to Xcode and Interface builder that these variables and methods can be used in Interface builder to link UI elements to our code.

*38.)Explain the difference between Generics and AnyObject in Swift.*

Generics are type safe, meaning if we pass a string as a generic and try to use as a integer the compiler will complain and we will not be able to compile our code. Because Swift is using **Static typing** and is able to give us a **compiler*error.***

If we use AnyObject, the compiler has no idea if the object can be treated as a String or as an Integer. It will allow us to do whatever we want with it.

39.)*Explain generics in Swift ?*

Generics create code that does not get specific about underlying data types. Generics allow us to know what type it is going to contain. Generics also provides optimization for our code.

*40.) What is the difference between Array vs NSArray ?*

Array can only hold one type of data, NSArray can hold different types of data. The array is value type, NSArray is immutable reference type.

41.) *What is Safe area ?*

**Safe area**allows us to create constraints to keep our content from being hidden by UIKit bars like status, navigation or tab bar. Previously we used topLayoutGuide and bottomLayoutGuide.

*42.)What is URLSession?*

When we want to retrieve contents from certain URL,we choose to use URLConnection. **There are three types of tasks:**

*(i.)Data tasks: getting data to memory*

*(ii.)Download tasks: downloading file to disk*

*(iii.)Upload tasks: uploading file from disk and receiving response as data in memory*

## 43.) *Explain* Implicitly Unwrapping *and Forced Unwrapping.*

## Implicitly Unwrapping

We define an Optional as impliticly unwrapped when we define its type like this:

let x: String!

This technique allows we to tell the compiler to automatically unwrap that value, as if it wasn't optional at all.

Simlarly to Type? which is a syntactic sugar for Optional<Type>, Type! is equivalent to ImplicitlyUnwrappedOptional<Type>.

A common example of implicitly unwrapped optionals - is how view controller define their IBOutlets:

@IBOutlet var messageLabel: UILabel!\*

@IBOutlet var actionButton: UIButton!

It makes sense to define the outlets as implicitly unwrapped optionals because these are going to be instantiated by Interface Builder. It would be cumbersome(hard) to always unwrap each view outlet inside view controllers.

Because of their implicitly unwrapped nature, if we forget to connect an outlet to it, Interface Builder view run the app then we'll get a runtime error when we try to access it in the view controllers.

Note : \*Above two coding lines doesn’t written with weak var, it means when we write weak var then memory can remove weak whenever they want but in case of only var , it’s a strong var and to remove strong var we have to remove it manually from the memory .

*Forced Unwrapping*

Force unwrap or force-unwrap or forced unwrapping, is the important technique in swift . It consist in adding a ! after an Optional value, to automatically unwrap it, without having to check whether it is nil or not.

let strings = ["mockcoding", "is", "a", "blog"]

let firstLength: Int = strings.first!.length

Like implicitly unwrapping, force unwrap uses a ! and makes the compiler treat an otherwise optional value as the type it wraps. Unlike implicitly unwrapping though, this technique is used on existing values.

We define a type as implicitly unwrapped, ImplicitlyUnwrappedOptional<T> and we force unwrap a value which has Optional type.

44.) *REST, HTTP, JSON — What’s that?*

**REST**, or REpresentational State Transfer, is a set of rules for designing consistent, easy-to-use and maintainable web APIs.

**HTTP** is the application protocol, or set of rules, web sites use to transfer data from the web server to client. The client (our web browser or app) use to indicate the desired action:

* **GET**: Used to retrieve data, such as a web page, but doesn’t alter any data on the server.
* **HEAD**: Identical to GET but only sends back the headers and none of the actual data.
* **POST**: Used to send data to the server, commonly used when filling a form and clicking submit.
* **PUT**: Used to send data to the specific location provided.
* **DELETE**: Deletes data from the specific location provided.

**JSON** stands for JavaScript Object Notation; it provides a straightforward, human-readable and portable mechanism for transporting data between two systems. Apple supplies the **JSONSerialization** class to help convert our objects in memory to JSON and vice-versa.

45.) *What is the difference between delegate and NSNotification?*

Delegates and NSNotifications can be used to accomplish nearly the same functionality. However, delegates are one-to-one while NSNotifications are one-to-many.

46*.) What is the difference between property and instance variable?*

A property is a more abstract concept. An instance variable is literally just a storage slot, like a slot in a struct. Normally other objects are never supposed to access them directly. Usually a property will return or set an instance variable, but it could use data from several or none at all.

47*.) Explain TypeCasting ?*

When we’re casting an object to another type in Swift , it’s pretty simple since there’s only one way to do it. In Swift, though, there are two ways to cast — one that’s safe and one that’s not .

* **as** used for upcasting and type casting to bridged type. //upcasting
* **as?** used for safe casting, return nil if failed.//downcasting
* **as!** used for force casting, crash if failed. Should only be used when we know the downcast will succeed.//downcasting

48.) *Why is everything in a do-catch block?*

In Swift, errors are thrown and handled inside of do-catch blocks.

49.) *What is UIStackView?*

UIStackView provides a way to layout a series of views horizontally or vertically. We can define how the contained views adjust themselves to the available space.

50.)*Explain difference between SDK and Framework ?*

SDK is a set of software development tools. This set is used for creation of applications. Framework is basically a platform which is used for developing software applications. It provides the necessary foundation on which the programs can be developed for a specific platform. SDK and Framework complement each other and SDKs are available for frameworks.

51.) *What is Enum ?*

Enum is a type that basically contains a group of related values in same umbrella.

52.)*What is bounding box?*

Bounding box is a term used in geometry; it refers to the smallest measure (area or volume) within which a given set of points.

53.)*What is the difference between strong, weak, read only and copy ?*

strong, weak, assign property attributes define how memory for that property will be managed.

**Strong** means that the reference count will be increased and  
the reference to it will be maintained through the life of the object.

**Weak ( non-strong reference )** means that we are pointing to an object but not increasing its reference count. It’s often used when creating a parent - child relationship. The parent has a strong reference to the child but the child only has a weak reference to the parent.

* Every time used on var
* Every time used on an optional type
* Automatically changes itself to nil

**Read only** means that we can set the property initially but then it can’t be changed.

**Copy** means that we’re copying the value of the object when it’s created. Also prevent its value from changing.

54.) *What is Dynamic Dispatch ?*

Dynamic Dispatch is the process of selecting which implementation of a polymorphic operation that’s a method or a function to call at run time. This means that when we want to invoke our methods like object method. But, Swift does not default to dynamic dispatch .

55.)*What is dynamic in Swift ?*

We use dynamic for subclasses of NSManagedObject. **dynamic** tells the compiler that getters and setters are implemented somewhere else.

56.) ***What is Singleton Pattern ?***

The Singleton design pattern ensures that only one instance exists for a given class and that -

there is a global access point to that instance. It usually uses lazy loading to create the single instance when it’s needed the first time.

57.)*Explain MVC* **.**

* **Model -**  responsible for the domain data or a data access layer which manipulates the data, think of ‘Person’ or ‘PersonDataProvider’ classes.
* **View -**  responsible for the presentation layer (GUI), for iOS environment, think of everything starting with ‘UI’ prefix.
* **Controller/Presenter/ViewModel -**  the glue or the mediator between the Model and the View, in general responsible for altering the Model by reacting to the user’s actions performed on the View and updating the View with changes from the Model.

58.)*What is JSON/PLIST limits ?*

* We create our objects and then serialized them to disk.
* It’s great and very limited use cases.
* We can’t obviously use complex queries to filter our results.
* It’s very slow.
* Each time we need something, we need to either serialize or deserialize it.
* it’s not thread-safe.

59.) *What is the main benefits of Swift ?*

The main **benefits** of Swift:

* Optional Types, which makes - applications crash-resistant
* Built-in error handling
* Closures
* Much/Most faster compared to other languages
* Type-safe language
* Supports pattern matching

60.) *Explain lazy in Swift ?*

An initial value of the lazy stored properties - is calculated only when the property is called for the first time. There are situations when the lazy properties come, very handy(convenient) to developers.

61.)*How to pass the variable as a reference ?*

We need to mention that there are two types of variables: reference and value type. The difference between these two types - is that by passing value type, the variable will create a copy of its data, and the reference type variable will just point to the original data in the memory.

62.) *Why it is better to use higher order functions?*

Functions that take another function as a parameter, or return a function, as a result, are known as higher-order functions. Swift defines these functions as CollectionTypes.

The very basic higher order function is a filter ,map and reduce.

*63.)Explain Higher Order Functions like Filter , Map and Reduce.*

Filter :

The filter method will return an array that has only elements that pass our filter specified in the closure.

let numbersAsStrings = numbers.filter(isIncluded: (Int) throws -> Bool)

Lets get all the numbers less than 5:

let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]

let numbersLessThanFive = numbers.filter { (a) -> Bool in

return a < 5

}

print(numbersLessThanFive)

o/p –

[0, 2, 1, 3, 4]

Inline syntax:

let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]

let numbersLessThanFive = numbers.filter { $0 < 5 }

print(numbersLessThanFive)

o/p –

[0, 2, 1, 3, 4]

Map :

Mapping is similar to sort in that it iterates through the array that is calling it, but instead of sorting it, changes each element of the array based on the closure passed to the method.

let numbersAsStrings = numbers.map(transform: (Int) throws -> T)

Notice that the return of the closure is **T**. Since an array is generic type and we are returning a new array, we can return an array with the type different than the starting array (if we so choose). Lets convert our numbers into strings.

let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]

let numbersAsStrings = numbers.map { (a) -> String in

return String(a)

}

print(numbersAsStrings)

o/p -

["0", "2", "1", "3", "6", "4", "9", "7", "8"]

If we read up on how to write inline syntax with closures we may write the same function this way.

let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]

let numbersAsStrings = numbers.map { String($0) }

print(numbersAsStrings)

o/p –

["0", "2", "1", "3", "6", "4", "9", "7", "8"]

Reduce :

The reduce function allows we to combine all the elements in an array and return an object of any type (generics)

let sumOfAllNumbers = numbers.reduce(initialResult: Result ,

nextPartialResult: (Result, Int) throws -> Result )

Notice that reduce has two parameters — initialResult and nextPartialResult. We need the initial result to tell us where to start, and the method that operates on that result based on the logic in the closure.

lets try turning all the numbers into one long string.

let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]

let sumOfAllNumbers = numbers.reduce("") { (result, a) -> String in

return result + String(a)

}

print(sumOfAllNumbers)

o/p –

021364978

Inline syntax:

let numbers: [Int] = [0, 2, 1, 3, 6, 4, 9, 7, 8]

let sumOfAllNumbers = numbers.reduce(“”) { $0 + String($1) }

print(sumOfAllNumbers)

o/p –

021364978

64.)*Explain Swift’s pattern matching techniques .*

* **Tuple patterns** are used to match values of corresponding tuple types.
* **Type-casting patterns** allow we to cast or match types.
* **Wildcard patterns** match and ignore any kind and type of value.
* **Optional patterns** are used to match optional values.
* **Enumeration case patterns** match cases of existing enumeration types.
* **Expression patterns** allow we to compare a given value against a given expression.

*65.)What are benefits of Guard ?*

There are two big benefits of guard. One is avoiding the pyramid of doom(complexity), as others have mentioned - lots of annoying “if let“ statements nested inside each other moving further and further to the right. The other benefit is provide an early exit out of the function using break or using return.

66.)*What is Functional programming ?*

There are three main concepts. These concepts are: separating functions and data, immutability, and first-class functions.

67.)*What are the limits of accessibility ?*

We can not use Dynamic Text with accessibility features.

68.)*What is ARC and how it is different from AutoRelease?*

Autorelease is still used ARC. ARC is used inside the scope, autorelease is used outside the scope of the function.

69.) *Explain differences between Foundation and CoreFoundation.*

The Foundation is a gathering of classes for running with numbers, strings, and collections. It also describes protocols, functions, data types and constants. CoreFoundation is a C-based alternative to Foundation. Foundation essentially is a CoreFoundation. We have a free bridge with NS counterpart.

70.)*What is intrinsic content size?*

Every view that contains content can calculate its intrinsic content size. The intrinsic content size is calculated by a method on every UIView instance. This method returns a CGSize instance.

71.)*Explain content extension.*

The content extension gives us the tools, we have in an app to design the notification.

72.)*What is Optional Binding ?*

We are going to take optional value and we are going to bind it non optional constant. We used If-let structure or Guard statement.

73.)*What is CocoaTouch ?*

CocoaTouch is a library used to build executable applications on iOS. CocoaTouch is an abstract layer on the iOS.

74.)*Explain Sequence in Swift.*

Sequence is a basic type in Swift for defining an aggregation of elements that distribute sequentially in a row. All collection types inherit from Sequence such as Array, Set, Dictionary.

75.)*Explain generic function zip(\_:\_:).*

According to the swift documentation, zip creates a sequence of pairs built out of two underlying. That means, we can create a dictionary includes two arrays.

76*.) What is an “app ID” and a “bundle ID” ?*

A bundle ID is the identifier of a single app. For example, if our organization’s domain is xyz.com and we create an app named Facebook, we could assign the string com.xyz.facebook as our app’s bundle ID.

An App ID is a two-part string used to identify one or more apps from a single development team. We need Apple Developer account for an App ID.

77.) ***W****hat’s the difference between accessibilityLabel and accessibilityIdentifier?*

accessibilityLabel is the value that’s read by VoiceOver to the end-user. As such, this should be a localized string. The text should also be capitalized. Because this helps with VoiceOver’s pronunciation. accessibilityLabel is used for testing and Visual Impaired users.

accessibilityIdentifier identifies an element via accessibility, but unlike accessibilityLabel, accessibilityIdentifier's purpose is purely to be used as an identifier for UI Automation tests. We use a value for testing process.

*78.)What is Responder Chain ?*

A ResponderChain is a hierarchy of objects that have the opportunity to respond to events received.

*79.)What is Regular expressions ?*

Regular expressions are special string patterns that describes how to search through a string.

*80.)What is Operator Overloading ?*

Operator overloading allows us to change how existing operators behave with types that both already exist.

81.)*What is Functions ?*

Functions let us group a series of statements together to perform some task. Once a function is created, it can be reused over and over in our code. If we find ourself repeating statements in our code, then a function may be the answer to avoid that repetition.

Good functions - accepts input and returns output. Bad functions set global variables and rely(trust) on other functions to work.

82.)*How do we download images?*

With URLSession, we can download an image as a data then convert it from NSData to UIImage, lastly we connect it to UIImageView IBOutlet. Better way is to use a [library](https://medium.com/ios-os-x-development/best-image-download-extension-library-for-swift-3-cf64ec1f84a0). Also with URLSession Adaptable Connectivity API we can built-in connectivity monitoring and run a request, if there is no connection. Request will wait and download whenever the resource is available instead of failing.

83.)*Explain differences between*[*WKWebView*](https://developer.apple.com/documentation/webkit/wkwebview?language=objc)*and UIWebView.*

WKWebView has own cookie storage and its not share by the whole app and all other web views like in the case of (UIWebView).

84.)*What is the difference b/w Layout Margins and Directional Layout Margins ?*

* **Layout Margins** - It is the property of a UIView is of type UIEdgeInsets and defines the top, left, bottom and right insets that when applied to the view’s frame define the view’s margin.
* **Directional Layout Margins -** That are aware of Right-To-Left (RTL) languages. This follows the pattern used when creating constraints with layout anchors.

*85.)What is the difference between Array vs NSArray ?*

Array can only hold one type of data, NSArray can hold different types of data. The array is value type whereas, NSArray is immutable reference type.

*86.)What is NSLayoutAnchor ?*

With iOS 9, Apple introduced the NSLayoutAnchor class to make writing autolayout easier with code.

There are three subclasses of **NSLayoutAnchor**:

* **NSLayoutXAxisAnchor:**This subclass is used to create horizontal constraints.
* **NSLayoutYAxisAnchor:**This subclass is used to create vertical constraints.
* **NSLayoutDimension:**This subclass is used to create width and height constraints.

*87.)What is circular dependencies ?*

A**circular dependency** is a relation between two or more modules which either directly or indirectly depend on each other to function properly. Such modules are also known as mutually recursive.

*88.)Explain Dependency Injection Container.*

The container keeps a map of each class type to an instance of that class. Then we can instantiate any class by simply providing the type to the container. Then the container automatically provides dependencies for that class.

*89.)Explain*[*ObjectMapper*](https://github.com/Hearst-DD/ObjectMapper)*for parsing JSON data.*

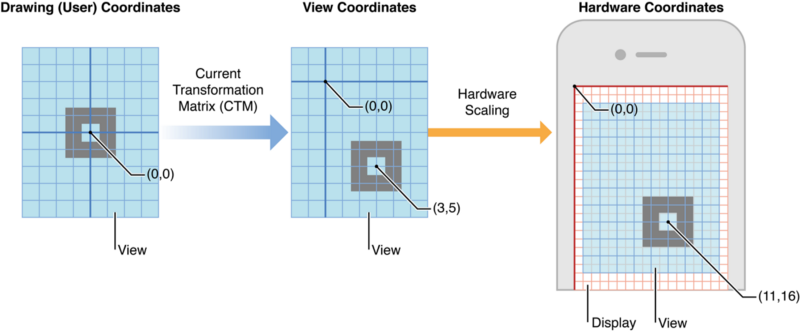
ObjectMapperconverts JSON data to strongly typed model objects. It has a two-way binding between JSON and deal with generic object and nested object. Also we can manage subclasses.

90.)*Explain Get Request Steps.*

We’re going to create the task. We’re going to receive the data back from the server and we’re going to handle it based on if there’s any errors and what the data is.

*91.)Explain coordinate systems in views.*

UIkit defines a default coordinate system with the origin in the top left corner, and axis extending to the right and down from the origin point. Views are laid out within this coordinate system to position and size them.



In addition to this Default Coordinate System, which we’ll call the Screen Coordinate System, an app’s window and views also define their own Local Coordinate System.

For example: single view is a view object tracks its size and location using a frame and bounds.

A frame is a rectangle, which specifies the size and location of the view within its SuperView coordinate system.

A bounds rectangle, specifies the size of the view within its own local coordinate system.

*92.)Explain*[*AlamoFire*](http://alamofire/)*Benefits.*

* AlamoFire creates a route. This means we can create the request and execute it to the server by one static function.
* AlamoFire provides method chaining for the requests that’s returned, which makes it easy for adding headers and handling responses.
* AlamoFire has multiple response handlers that’s returned in binary form, text, parse JSON and we can even use multiple of these for a given request.
* AlamoFire has the method chaining allows for response validation. We can call validation to check for the status code of the HTTP response, the content type, or any custom validation we might need to do for our app.
* AlamoFire gives us that use a couple of protocols, URLConvertible and URLRequestConvertible. These protocols can be passed in when creating a request.
* AlamoFire provides extensions can be passed in to create the request.

*93.) Explain any three*[*SwiftyJSON*](http://alamofire/)*Benefits.*

* **SwiftyJson** is a library used to easily deal with **JSON Data** in swift.
* SwiftyJSON is a library to elegantly(nicely) serialize JSON data into Swift objects. We don't need to use Swifty JSON to serialize JSON data in swift, but it can get problems having to deal with optionals.
* SwiftyJSON is a super-simplified JSON parsing library that gives us clearer syntax compared to built-in iOS libraries even more than JSONEncoder from Codable.

*94.)Explain Classes are a Reference Type.*

In all programming languages, classes are the reference types which means that we can’t store the actual value of class, we only store its reference. Classes use heap memory rather than stack to store the reference of variables. Heap is large pool of memory comparatively with stack memory.

  Classes use the heap Memory which mean that when we initialize class with its instance, the value will store in heap memory.

  Following is the example of showing how classes are reference types in swift.

class UserDetails {

var name: String

init(name: String) {

self.name = name

}

}

var uinfo1 = UserDetails(name: "Ranjeet")

var uinfo2 = uinfo1

uinfo2.name = "Sujeet"

print(uinfo1.name)

print(uinfo2.name)

If we observe above example , we defined a class (**UserDetails**) and created an instance for the class and assigned to a variable “**uinfo1**”. After that again we created a new variable “**uinfo2**” and assigned previously created variable “**uinfo1**” to new variable.

Initially we updated name in variable “**uinfo1**” after that we assign variable “**uinfo1**” to “**uinfo2**” and again we updated name value.

Whenever we update the value in one place same will reflect for all the references in application because classes are reference types.

When we run above program , we will get result like as shown below

Sujeet

Sujeet

This is how classes will behave as a reference types in swift .

*95.)What is the difference beteen Identical to Operators and Not Identical to Operators.*

|  |  |
| --- | --- |
| Identical to Operators | Not Identical to Operators |
| Operator used is (===) | Operator used is (!==) |
| Returns true when two constants or variables pointing to a same instance | Returns true when two constants or variables pointing to a different instance |

*96.)Explain Structures are a Value Type .*

In swift, structures are the value type so the property values will always copy when we assign to another variable and the original value will remain same and does not affect the original value of variable.

Following is the example of showing how structures are value types in swift.

struct UserDetails {

var name: String = ""

}

var userInfo = UserDetails(name: "Ranjeet")

var info2 = userInfo

info2.name = "Sujeet"

print(info2.name)

print(userInfo.name)

If we observe above example , we defined structure(**UserDetails**) and created instance for the structure and assigned to a variable “**userInfo**”. After that again we created new variable “**info2**” and assigned previously created variable “**userInfo**” to new variable.

Initially we updated name in variable “**userInfo**” and once we assign variable “**userInfo**” to “**info2**” again we updated name value but the value of variable “**info2**” will not affect “**userInfo**” value because structures are value types so when we pass value to variable the original value of variable will be remain same.

When we run above program , we will get result like as shown below

Sujeet

Ranjeet

*97.) What is the difference between Classes and Structures.*

In swift, [classes](https://www.tutlane.com/tutorial/swift/swift-classes) have some of additional capabilities when compared with structures. Following are the differences between [classes](https://www.tutlane.com/tutorial/swift/swift-classes) and structures .

* No inheritance in Structures, it’s not possible for us to Inherit from other types.
* [Classes](https://www.tutlane.com/tutorial/swift/swift-classes) use **type-casting** in swift to define a class as a superclass or subclass but **type-casting** not available for structures.
* By default no mutation available for structure methods. In case if we try to change the value of property, compiler will throw an error.
* Deinitializers not available in structures to free up assigned resources in instance.
* Not possible for us to create multiple references to the same instance in Structures.

Note: Object is the instance of a class.

*98.)How to choose Between Classes and Structures.*

In swift both structures and [classes](https://www.tutlane.com/tutorial/swift/swift-classes) is having their own priorities so we need to consider following things to decide whether to use class or structure.

* In swift if we want to share the mutable state between different parts of application then we need to use a class.
* Structures are significantly faster, use less memory and useful for few simple data values.
* In structures **inheritance** will not support so if we want to implement inheritance functionality we need to go for classes.

*99.)What is the mutating keyword ?*

The **mutating** keyword is used to let variables be modified in a **struct** or **enum**.

100.)what is reloadData() in tableView and collectionView.

reloadData() for tableView

Reloads the rows and sections of the table view.

func reloadData()

Call this method to reload all the data that is used to construct the table, including cells, section headers and footers, index arrays and so on. For efficiency, the table view redisplays only those rows that are visible. It adjust offsets if the table shrinks as a result of the reload. The table view’s delegate or data source calls this method when it wants the table view to completely reload its data.

reloadData() for collectionView

Reloads all of the data for the collection view.

func reloadData()

Call this method sparingly(Kami ke saath) when we need to reload all of the items in the collection view. This causes the collection view to discard any currently visible items (including placeholders) and recreate items based on the current state of the data source object. For efficiency, the collection view only displays those cells and supplementary views that are visible. If the collection data shrinks as a result of the reload, the collection view adjust its scrolling offsets accordingly.

101.) How to initialize the dictionary ?

var addEmployee = Dictionary<String, String>()

102.)Write one example for Dictionary creation .

addEmployee["FName"] = txtFName.text!  
addEmployee["LName"] = txtLName.text!  
addEmployee["EmailID"] = txtEmailID.text!  
addEmployee["DOB"] = dateField.text!

Another Way,

var addEmployee:[String:String] = ["FName":"FName" , "LName":"LName" , "EmailID":"EmailID" , "DOB":"DOB"]

103.)How to add Dictionary to Array?

var employeeList = Array<Dictionary<String,String>>()//initialized array

104.)Differentiate between Local Variable and Global Variable.

|  |  |
| --- | --- |
| **Local Variables** | **Global Variables** |
| Variables that are defined within a function, method or closure context. | Variables that are defined outside function, method, closure or type context. |
| Used to store and retrieve values. | Used to store and retrieve values. |
| Stored properties is used to get and set the values. | Stored properties is used to get and set the values. |
| Computed properties are also used. | Computed properties are also used. |

105.)Explain do , try , catch , throw and throws in brief.

## Error Handling: Error Handling is all about handling the failing conditions gracefully.

An error can lead to runtime errors or changes in the flow of the program.

We come across different kinds of errors in our projects:

* Logic Errors
* Type Conversion Errors
* External Errors such as FileNotFound etc.

The brute(hard) force way to handle errors - is by using **if else statements** where we check each and every possible error. But this can lead to bloated codes(phula hua) with too many nested conditions.

Errors are just values of a certain type. Swift does not support checked exceptions.

Error Protocol

Error Protocol is just a type for representing error values that can be thrown.

Swift requires we to create a custom Error type. Typically an [Enum](https://www.journaldev.com/15343/swift-enum) is used which conforms to the **Error** [Protocol](https://www.journaldev.com/16803/swift-protocol).

The Error Protocol is more or less empty. Hence we don’t need to override anything from them. Error Protocol is must for Error Handling and creating Error types.

Let’s create a basic enum which conforms to this Error Protocol.

enum UserDetailError: Error {

case noValidName

case noValidAge

}

Now let’s use this Error Type in our classes and functions.

throws and throw

If a function or an initializer can throw an error, the throws modifier must be added in the definition itself right after the parentheses and just before the return type.

func userTest() throws -> <Return Type> {

}

The throws keyword would propagate the error from the function to the calling code.  
Otherwise, a non-throwing function must handle the error inside that function’s code itself.

throw keyword is used for throwing errors from the error type defined.

Let’s look at an example demonstrating throws and throw in a function:

func userTest(age: Int, name: String) throws {

guard age > 0 else{

throw UserDetailError.noValidAge

}

guard name.count > 0 else{

throw UserDetailError.noValidName

}

}

In Error Handling, guard let is useful in the sense that we can replace the return statement in the else block with the throwing error. This prevents too many if else conditions.  
Let’s look at it with the example below.

func userTest(age: Int, name: String) throws {

guard age > 0 else{

throw UserDetailError.noValidAge

}

guard name.count > 0 else{

throw UserDetailError.noValidName

}

}

Note: We cannot add the Error type after the throws keyword in Swift.

In the above code, if the condition in the guard let fails, it’ll throw an error and the function would return there itself.  
Let’s look at how to handle these errors.

### Swift try, do-catch

In Swift, contrary(against) to Java, do-catch block is used to handle errors in place of try-catch.

Every function that has throws needs to set in the try statement since it has a potential error.

Swift try statement is executed only when it is inside the do-catch block as shown below.

do{

try userTest(age: -1, name: "")

} catch let error {

print("Error: \(error)")

}

Below code shows the output of above program.

enum UserDetailError: Error{

case noValidName

case noValidAge

}

func userTest(age: Int, name: String) throws{

guard age > 0 else{

throw UserDetailError.noValidAge

}

guard name.count > 0 else{

throw UserDetailError.noValidName

}

}

do{

try userTest(age: -1, name: "")

}

catch let error{

print("Error: \(error)")

}

o/p :

**Error: noValidAge**

Alternatively we can do this:

do{

try userTest(age: -1, name: "")

}

catch UserDetailError.noValidName

{

print("The name isn't valid")

}

catch UserDetailError.noValidAge

{

print("The age isn't valid")

}

catch let error {

print("Unspecified Error: \(error)")

}

### Throwing Errors in Initializers

We can add throws in the initializer in the following way.

enum StudentError: Error {

case invalid(String)

case tooShort

}

class Student {

var name: String?

init(name: String?) throws {

guard let name = name else{

throw StudentError.invalid("Invalid")

}

self.name = name

}

func myName(str: String) throws -> String {

guard str.count > 5

else{

throw StudentError.tooShort

}

return "My name is \(str)"

}

}

Now to initialize the class we normally do the following, right?

var s = Student(name: nil)

//compiler error

**WRONG**

Since the intializer is throwing errors we need to append try keyword as shown below.

do{

var s = try Student(name: nil)

}

catch let error

{

print(error)

}

//prints

//invalid("Invalid")

Let’s call the class function on the object too as shown below.

enum StudentError: Error{

case invalid(String)

case tooShort

}

class Student{

var name: String?

init(name: String?) throws{

guard let name = name else{

throw StudentError.invalid("Invalid")}

self.name = name

}

func myName(str: String) throws -> String{

guard str.count > 5 else{

throw StudentError.tooShort

}

return "My name is \(str)"

}

}

do{

var s = try Student(name: "Anupam")

try s.myName(str: "Anup")

}

catch let error{

print(error)

}

o/p –

tooShort

### Swift try, try? and try!

* Swift try is the most basic way to dealing with functions that can throw errors.
* try? is used to handle errors by converting the error into an optional value. This way if an error occurs, the function would return a nil and we known Optionals can be nil in Swift. Hence for try? we can get rid of do-catch block.
* try! is used to assert that the error won’t occur. Should be only used when we’re absolutely sure that the function won’t throw an error. Like try?, try! works without a do-catch block.

var t1 = try? Student(name: nil)

var t2 = try! Student(name: "Anupam")