Ques1: explain the functionality of drag and drop in interaction?

Answer- With drag and drop in iOS, users can quickly and easily move text, images, and files from one app to another. Simply tap and hold to pick up the content and drag it to the other app or file. A drag-and-drop activity can take place in a single app, or it can start in one app and end in another. The app from which an item is dragged is called the source app. The app on which an item is dropped is called the destination app. Drag and drop takes advantage of the power of Multi-Touch, letting you to select multiple items just by tapping and then drag it.

Ques2: Write a short note on table view?

Answer- A table view displays a list of items in a single column which is specifically design to display a scrolling list of items and allows user to scroll through the table(by UIScrollview). iOS comes with a built-in class, UITableView and table view is managed by a table view controller (UITableViewController).

Most apps in some ways make use of table view to display list of data. Example : Mail app uses table view to display your mail boxes and emails.

Ques3: explain usage and functionality of keyboard with the help of example?

Answer- You use text fields to gather text-based input from the user using the onscreen keyboard. The keyboard is configurable for many different types of input such as plain text, emails, numbers, and so on. Text fields use the target-action mechanism and a delegate object to report changes made during the course of editing.

In addition to its basic text-editing behavior, you can add overlay views to a text field to display additional information and provide additional tappable controls. You might add custom overlay views for elements such as a bookmarks button or search icon. Text fields provide a built-in overlay view to clear the current text. The use of custom overlay views is optional.

Eg: Whenever the user taps in an object capable of accepting text input, the object asks the system to display an appropriate keyboard. Depending on the needs of your program and the user's preferred language, the system might display one of several different keyboards. Although your app cannot control the user's preferred

language (and thus the keyboard's input method), it can control attributes of the keyboard that indicate its intended use, such as the configuration of any special keys and its behaviors

Ques4: Explain the file system along with core data and UI document?

Answer- File system is one of the fundamental resources used by all processes. As, File system handles the persistent storage of data files, apps, and the files associated with the operating system itself. APFS is the default file system in macOS, iOS, watchOS, and tvOS.

Core Data is one of the most popular frameworks provided by apple for ios. It is used to manage the model layer object in our application to save, track, modify and filter the data within iOS apps. Core data provides an in memory persistent store and three disk based persistent stores i.e XML (atomic), Binary (atomic), Sqlite and in –memory.

UI document in ios comes under UIDocument class which designed to provide an easy to use interface for the creation and management of documents and content. While primarily intended to ease the process of storing files using iCloud, UIDocument also provides additional benefits in terms of file handling on the local file system such as reading and writing data asynchronously on a background queue, handling of version conflicts on a file.

Ques5: explain the functionality of alert and action sheet? Code

Answer- Alerts convey important information related to the state of your app or the device, and often request feedback. An alert consists of a title, an optional message, one or more buttons, and optional text fields for gathering input. It gernally pop up in between the screen.

Code: Display button to display alert

```
@IBAction func showAlert(_ sender: Any) {
    let alertController = UIAlertController(title: "iOScreator", message:
        "Hello, world!", preferredStyle: UIAlertControllerStyle.alert)
    alertController.addAction(UIAlertAction(title: "Dismiss", style:
UIAlertActionStyle.default,handler: nil))
    self.present(alertController, animated: true, completion: nil)
}
```

An action sheet is a specific style of alert that appears in response to a control or action, and presents a set of two or more choices related to the current context. Use an action sheet to let people initiate tasks, or to request confirmation before performing a potentially destructive operation. It gernally slides from the bottom of the screen.

Code:

```
@IBAction func displayActionSheet(_ sender: Any) {
    let optionMenu = UIAlertController(title: nil, message: "Choose Option",
preferredStyle: .actionSheet)
    let deleteAction = UIAlertAction(title: "Delete", style: .default)
    let saveAction = UIAlertAction(title: "Save", style: .default)
    let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)
    optionMenu.addAction(deleteAction)
    optionMenu.addAction(saveAction)
    optionMenu.addAction(cancelAction)
    self.present(optionMenu, animated: true, completion: nil)
}
```

Ques6: Explain functionality of notification? Code

Answer- Apple Push Notification service (APNs) is the centerpiece of the remote notifications feature. It is a robust, secure, and highly efficient service for app developers to propagate information to iOS. Notifications are great for keeping users informed with timely and relevant content, whether your app is running in the background or inactive, notifications can display a message etc. For example, notifications may occur when a message has arrived, an event is about to occur, new data is available, or the status of something has changed.

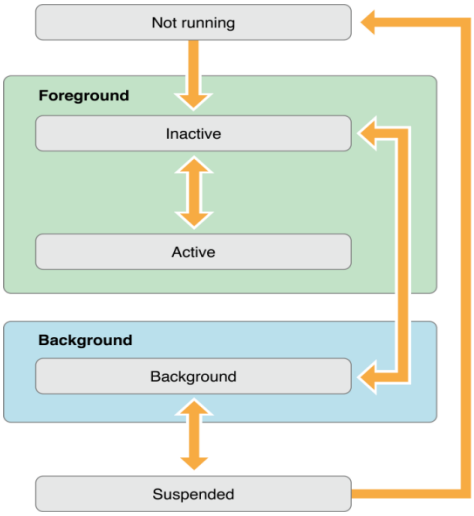
Code:

```
Import usernotification

func registerForPushNotifications() {

    UNUserNotificationCenter.current().delegate = self

    UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .sound, .badge]) {

        (granted, error) in

        print("Permission granted: \(granted)")

        // 1. Check if permission granted

        guard granted else { return }

        // 2. Attempt registration for remote notifications on the main thread

        DispatchQueue.main.async {

            UIApplication.shared.registerForRemoteNotifications()

        } }   }

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
```

```
    registerForPushNotifications()
    return true
}
```

Ques7: Write a short note on application life cycle with diagram?

Answer-



Not running: The app has not been launched or was running but was terminated by the system.

Inactive: The app is running in the foreground but is currently not receiving events. An app usually stays in this state only briefly as it transitions to a different state.

Active: The app is running in the foreground and is receiving events. This is the normal mode for foreground apps.

Background: The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended.

Suspended: The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code. When

a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.

To perform those operation some functions are use:

application:willFinishLaunchingWithOptions:—This method is your app's first chance to execute code at launch time.

application:didFinishLaunchingWithOptions:—This method allows you to perform any final initialization before your app is displayed to the user.

applicationDidBecomeActive:—Lets your app know that it is about to become the foreground app. Use this method for any last minute preparation.

applicationWillResignActive:—Lets you know that your app is transitioning away from being the foreground app. Use this method to put your app into a quiescent state.

applicationDidEnterBackground:—Lets you know that your app is now running in the background and may be suspended at any time.

applicationWillEnterForeground:—Lets you know that your app is moving out of the background and back into the foreground, but that it is not yet active.

applicationWillTerminate:—Lets you know that your app is being terminated. This method is not called if your app is suspended.

Ques8: Explain core motion with the help of example?

Answer-

Core Motion is a Framework provided by Apple which gives us the possibility to easily access the data of the motion and movement of the device. Core Motion reports motion and environment related data from the onboard hardware of iOS devices, including from the accelerometers (instantaneous acceleration of the device in three dimensional space) and gyroscopes (instantaneous rotation around the device's three primary axes.), and magnetometer (indicating the device's orientation relative). We use this framework to access hardware-generated data so that you can use it in your app.

For example, a game might use accelerometer and gyroscope data to control onscreen game behavior.

Ques9: compare core image and vision?

Answer-

Core Image is an image processing and analysis technology designed to provide high-performance processing for still and video images. It operates on image data types from the Core Graphics, Core Video, and Image I/O frameworks, using either a GPU

The Core Image framework provides:

- Access to built-in image processing filters
- Feature detection capability
- Support for automatic image enhancement
- The ability to chain multiple filters together to create custom effects
- Support for creating custom filters that run on a GPU
- Core Image is Efficient and Easy to Use for Processing and Analyzing Images: Core Image provides hundreds of built-in filters. You set up filters by supplying key-value pairs for a filter’s input parameters. The output of one filter can be the input of another, making it possible to chain numerous filters together to create amazing effects.

Vision is a high-level framework that provides an easy to use API for handling many computer vision tasks. It performs face and face landmark detection, text detection, barcode recognition, image registration, and general feature tracking.

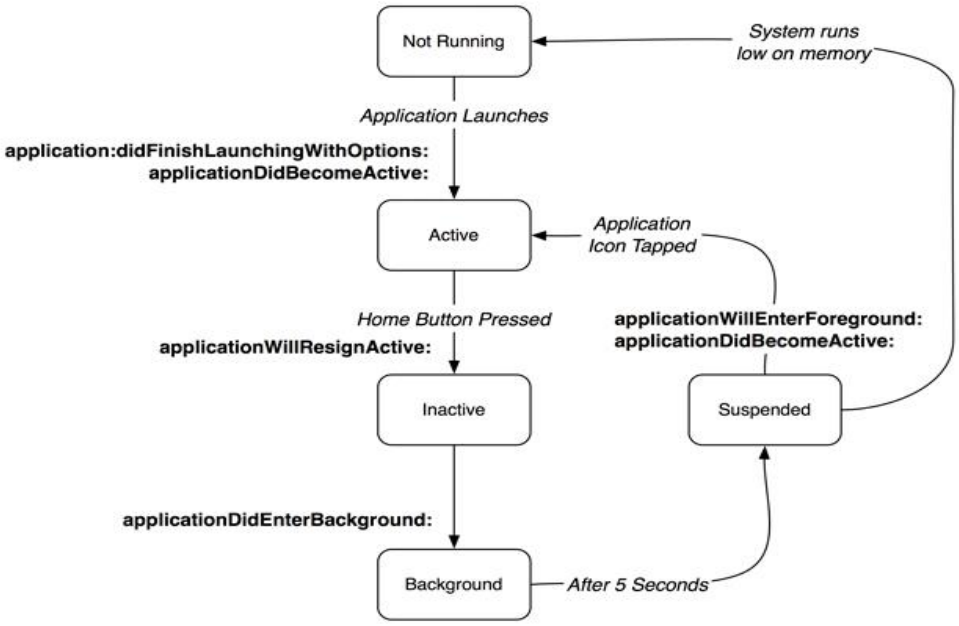
Vision framework allows you to:

- Detect face rectangle and face landmarks (face contour, median line, eyes, brows, nose, lips, pupils position)
- Find projected rectangular regions surface.
- Find and recognizes barcodes.
- Find regions of visible text.
- Determine the horizon angle in an image.
- Detect transforms needed to align the content of two images.
- Process images with Core ML model.
- Track movement of a previously identified arbitrary object across multiple images or video frames.

Ques10: How you can response to the state transition in your app? Explain with example?

Answer-

Same as ques 7



Q.11) Explain usage of controller object in UIapplication?

Answer-

Controller object UIApplication is used without subclassing to manage the application event loop. It coordinates other high-level app behaviors.

Some of the behaviours of UIApplications:

- It works along with the app delegate object which contains app-level logic.
- Control the app's response to changes in interface orientation.
- Register for remote notifications.
- Extend the execution of the app so that it can finish a task in the background.

- Schedule and cancel local notifications.

- Coordinate the reception of remote-control events.

(A major role of a UIApplication object is to handle the initial routing of incoming user events. It also dispatches action messages forwarded to it by control objects (UIControl) to the appropriate target objects.)

Q.12)

(a)State and compare categories and extensions ?

Category in Objective-C, Extension in Swift are the same concept. They can help you to organise your class code. With category and extension, you don't have to create a inherited class.

Categories allow you to add methods outside of the main interface file. Whereas Extensions must be implemented in main Interface file. Which means we can safely conclude you cannot use extensions for extending Builtin classes or Classes for which you don't have the source code, there you should use Categories. And to use extensions you need access to the source of the class you are extending.

(b) State and compare KVO and KVC ?

KVC is a form of coding that allows you to access an object's properties indirectly, using strings to access them instead of the property's accessors or instead of accessing the variables directly. To enable such mechanism, your classes must comply to the NSKeyValueCoding informal protocol.

Nevertheless, there is a much better way to observe for changes in properties, and Apple uses it a lot in its own software. That way is not used a lot by programmers because it seems hard to be learnt, however this is not true. Once you get to know it, you'll probably love it and you'll see that it's much effortless and easier to track down changes on single properties or collections, such as arrays. This method is called **Key-Value Observing**, but is mostly known as **KVO**.

(c) NSNotification and Delegate?

Delegation is a type of design pattern that, as its name
suggests, delegatesresponsibility and control from one class to another for communication purposes. Basically one class (the delegator) states some behavior

that the other class (or delegate) must comply with. The delegator can thus count on that behavior being carried out every time the delegate is used. Additionally, the delegate can return information.

Apple describes NSNotificationCenter as “a mechanism for broadcasting information within a program.” NSNotificationCenter sends out a notification, and observers are set up to “listen” for it and then take their own specified actions. NSNotificationCenter provides a way to inform multiple objects of an event. (Note that it also needs to be removed once observers no longer need to listen for a notification.)

Unlike the delegation phone call analogy, NSNotificationCenter is more like a radio broadcast: multiple receivers can tune in and hear the message, but cannot relay information back to the sender.

Q.13) What is ARC?

Answer- Swift uses Automatic Reference Counting (ARC) to track and manage your app’s memory usage. In most cases, this means that memory management “just works” in Swift, and you do not need to think about memory management yourself. ARC automatically frees up the memory used by class instances when those instances are no longer needed.

Every time you create a new instance of a class, ARC allocates a chunk of memory to store information about that instance. This memory holds information about the type of the instance, together with the values of any stored properties associated with that instance.

Additionally, when an instance is no longer needed, ARC frees up the memory used by that instance so that the memory can be used for other purposes instead. This ensures that class instances do not take up space in memory when they are no longer needed.

However, if ARC were to deallocate an instance that was still in use, it would no longer be possible to access that instance’s properties, or call that instance’s methods. Indeed, if you tried to access the instance, your app would most likely crash.

To make sure that instances don't disappear while they are still needed, ARC tracks how many properties, constants, and variables are currently referring to each class instance. ARC will not deallocate an instance as long as at least one active reference to that instance still exists.

To make this possible, whenever you assign a class instance to a property, constant, or variable, that property, constant, or variable makes a strong reference to the instance. The reference is called a "strong" reference because it keeps a firm hold on that instance, and does not allow it to be deallocated for as long as that strong reference remains.

Q.14) How do you find memory leaks in ARC.What is use of NSoperation?

Answer- In Swift, memory leaks exist with retain cycles. A retain cycle occurs when two objects hold **strong** references to each other.

By using -
1. Static analyzer.

static analyzers are able to find simple cases of memory leaks. But in practice you have the memory leaks mostly when the code is complicated and the memory is free/allocated in different parts of the program. Therefore, static analysis is really not very efficient.

2. Instrument

The Instruments tool provided with Xcode can give you valuable insight into the lifetime of objects used throughout your application. To get started, connect your testing device.

Configure the information that Instruments should capture:

- **Discard unrecorded data upon stop:** Instruments tends to capture more information than it can process in real time. Uncheck this option so that it will finish processing any queued raw data when you hit Stop.

- **Discard events for freed memory:** Enabling this will make it much easier to navigate the data that Instruments captures. We're trying to find leaked memory, so information about objects that have been allocated and released correctly isn't very interesting.
- **Record reference counts:** In our analysis, an object that is still listed will have a reference count greater than one, so the specific value is not all that important. But it can be interesting to go back later and see where an object was retained and released. Leave this option unchecked for now since it can really slow down your app.
- **Recorded types:** If you already have some idea of which objects are leaking, you can filter the recorded types here. At a minimum, you should check the options to ignore types with the prefixes "NS," "CF," and "malloc," since these are all iOS-specific or low-level functions. (These types of objects may leak, but typically it's because something else at a higher level leaked them first.) Reducing the amount of information that Instruments has to capture and process will keep your app from bogging down too much while Instruments is attached.

NSOperation

An abstract class that represents the code and data associated with a single task. Because the NSOperation class is an abstract class, you do not use it directly but instead subclass or use one of the system-defined subclasses (NSInvocationOperation or NSBlockOperation) to perform the actual task. Despite being abstract, the base implementation of NSOperation does include significant logic to coordinate the safe execution of your task.

The presence of this built-in logic allows you to focus on the actual implementation of your task, rather than on the glue code needed to ensure it works correctly with other system objects.

Q.15 What is GCD ?What are its advantages over NS thread?

Answer- Grand Central Dispatch (GCD) is a low-level API for managing concurrent operations. It can help you improve your app's responsiveness by

deferring computationally expensive tasks to the background. It’s an easier concurrency model to work with than locks and threads.

GCD provides three main types of queues:

1. **Main queue**: runs on the main thread and is a serial queue.
2. **Global queues**: concurrent queues that are shared by the whole system. There are four such queues with different priorities : high, default, low, and background. The background priority queue has the lowest priority and is throttled in any I/O activity to minimize negative system impact.
3. **Custom queues**: queues that you create which can be serial or concurrent. Requests in these queues actually end up in one of the global queues.
4. With GCD, you can dispatch a task either synchronously or asynchronously.
5. A synchronous function returns control to the caller after the task completes. You can schedule a unit of work synchronously by calling `DispatchQueue.sync(execute:)`.
6. An asynchronous function returns immediately, ordering the task to start but not waiting for it to complete. Thus, an asynchronous function does not block the current thread of execution from proceeding on to the next function. You can schedule a unit of work asynchronously by calling `DispatchQueue.async(execute:)`.

NSThread-

Use NSThread when you want a unit of computational work done without necessarily either waiting for other units to finish, or holding up other computational work.
You can put almost any work into a thread, if it is sensible to do so.

A good example is a network request, where you set up a thread to download data from, say, a web server. Your thread will fire a "handler" function when it has completed its work. The handler works with the downloaded data; for example, parsing XML data from a web service.

You would use a thread in this example, because you don't want the entire application to lock up while your app downloads data over the network and processes it. An `NSThread` instance puts this unit of work into its own little "space" that allows the larger app to continue to interact with the user.

Q.16> How will you find memory leaks in MRC?

Answer- A memory leak is a portion of memory that is occupied forever and never used again. It is garbage that takes space and causes problems. One of the most common problems that generate memory leaks in iOS is retain cycles. This occurs when we make circular references between two or more objects. MRC stands for Manual Reference Counting. We can find

1. Static analyzer- The static analyzer tries out thousands of possible code paths in a few seconds, reporting potential bugs that might have remained hidden or bugs that might be nearly impossible to replicate. To perform static code analysis, choose Product > Analyze. The Xcode static analyzer parses the project source code and identifies these types of problems:

1. Logic flaws, such as accessing uninitialized variables and dereferencing null pointers
2. Memory management flaws, such as leaking allocated memory
3. Dead store (unused variable) flaws

2. Instrument- It is a powerful and flexible performance analysis and testing tool that's part of the Xcode toolset. It's designed to help you profile your OS X and iOS apps, processes, and devices in order to better understand and optimize their behavior and performance.

Q.17> What is the use of NS operation?

Answer- The NS Operation class is an abstract class you use to encapsulate the code and data associated with a single task. Because it is abstract, you do not use this class directly but instead subclass or use one of the system-defined subclasses (NS Invocation Operation or Block Operation) to perform the actual task. NS Operation represents a single unit of work.

Q.18> What is GCD?

Answer- GCD stands for Grand Central Dispatch. GCD or Grand Central Dispatch is used to optimize applications for multi-core processors. It basically dispatches the thread pool management from the developer to the OS, helping in concurrent code execution. It enables very simple use of a task-based concurrency model. GCD offered a number of benefits:

1. Improving app responsiveness by running complex tasks in the background.
2. Providing a concurrency model better than the usual locks and threads to prevent concurrency bugs.

Q.19> What are the advantages over NS threads?

Answer- There are many reasons for using NS Thread in your application based on the architecture and complexity of the application. The main purpose to use threads in application, is concurrency and asynchronously working for process and send notification message to main thread. By this way, developer doesn't need to block main thread while applications are executing different tasks. Use NS Thread when you want or need to have direct control over the threads you create.

Q.20> Explain the difference between atomic and non-atomic synthesis properties?

Answer-

Atomic

- is the default behavior
- will ensure the present process is completed by the CPU, before another process accesses the variable
- is not fast, as it ensures the process is completed entirely
- use if the instance variable is going to be accessed in a multithreaded environment.

Non-Atomic

- is NOT the default behavior
- faster (for synthesized code, that is, for variables created using @property and @synthesize)
- not thread-safe
- may result in unexpected behavior, when two different process access the same variable at the same time
- If the instance variable is not going to be changed by multiple threads you can use it. It improves the performance.

Q.21) What is toll free bridging and when is it used?

Answer-

There are a number of data types in the Core Foundation framework and the Foundation framework that can be used interchangeably. This capability, called toll-free bridging, means that you can use the same data type as the parameter to a Core Foundation function call or as the receiver of an Objective-C message. For example, NSLocale (see NSLocale Class Reference) is interchangeable with its Core Foundation counterpart, CFLocale (see CFLocale Reference). Therefore, in a method where you see an NSLocale * parameter, you can pass a CFLocaleRef, and in a function where you see a CFLocaleRef parameter, you can pass an NSLocale instance.

Q.22) What is the extension, process and thread explain with example.

Answer-

Extensions add new functionality to an existing class, structure, enumeration, or protocol type. This includes the ability to extend types for which you do not have access to the original source code (known as retroactive modeling). Extensions are similar to categories in Objective-C. (Unlike Objective-C categories, Swift extensions do not have names.)

Extensions in Swift can:

- Add computed instance properties and computed type properties

- Define instance methods and type methods
- Provide new initializers
- Define subscripts
- Define and use new nested types
- Make an existing type conform to a protocol

Example---

Extension are pretty easy to create by writing outside the your class or Controller class like extension and your class name

```
extension UserModel {
    //your functionality
}
```

Process

A process is a running instance of an application. Processes are the (binary) executable packages. Usually in your program code you define tasks to execute. Each process contains the current activity state of his own.

A process is a self contained execution environment and it can be seen as a program or application. However a program itself contains multiple processes inside it. Java runtime environment runs as a single process which contains different classes and programs as processes.

Thread

Thread can be called lightweight process. Thread requires less resources to create and exists in the process, thread shares the process resources.

A thread is an independent set of values for the processor registers (for a single core). Since this includes the Instruction Pointer (aka Program Counter), it controls what executes in what order. It also includes the Stack Pointer, which had better point to a unique area of memory for each thread or else they will interfere with each other."

Example-

```
Thread t1 = new Thread(new MyClass ());
t1.start();
```

Q.23)how will you make snippet thread safe?

Answer-

7 Techniques for Thread-Safe Classes

- **No State-** When multiple threads access the same instance or static variable, you must somehow coordinate the access to this variable. The easiest way to do this is simply by avoiding instance or static variables.

```
public static int subtractExact(int x, int y) {
    int r = x - y;
    if (((x ^ y) & (x ^ r)) < 0) {
        throw new ArithmeticException("integer overflow");
    }
    return r;
}
```

- **No Shared State**- If you cannot avoid state, do not share the state. The state should only be owned by a single thread.

- **Message Passing-** A technique to do this is by passing messages between threads. You can implement message passing using a concurrent queue from the package java.util.concurrent. Or, better yet, use a framework like Akka, a framework for actor style concurrency. The following example shows how to send a message with

  Akka: target.tell(message, getSelf());

- **Immutable State-**To avoid the problem where a sending thread changes the message when the message is read by another thread, messages should be immutable. The Akka framework, therefore, has the convention that all messages have to be immutable
- **Use the Data Structures From java.util.concurrent**-Message passing uses concurrent queues for communication between threads. Concurrent queues are one of the data structures provided in the package java.util.concurrent. This package provides classes for concurrent maps, queues, dequeues, sets, and lists. Those data structures are highly optimized and tested for thread safety.

- **Synchronized Blocks-** If you cannot use one of the above techniques, use synchronized locks. By putting a lock inside a synchronized block, you make sure that only one thread at a time can execute this section.

```
synchronized(lock)
{
   i++;
}
```

- **Volatile Fields-**Normal, nonvolatile fields can be cached in registers or caches. Through the declaration of a variable as volatile, you tell the JVM and the compiler to always return the latest written value. This not only applies to the variable itself, but to all values written by the thread that has written to the volatile field. The following shows an example of a volatile instance variable:

```
public class ExampleVolatileField
{
   private volatile int  volatileField;
}
```

Q.24) What happen when you create a block?

Answer- Blocks are a language-level feature added to C, Objective-C and C++, which allow you to create distinct segments of code that can be passed around to methods or functions as if they were values. Blocks are Objective-C objects, which means they can be added to collections like NSArray or NSDictionary. They also have the ability to capture values from the enclosing scope, making them similar to closures or lambdas in other programming languages.

- **ReturnType**: Any data type Objective-C supports, or void if the block returns nothing.
- **^blockName**: Always remember that the block name is preceded by the ^ symbol. The block name can be any string you like, just like you name any other variable or method. Remember that both the ^ and the block name are enclosed in parentheses ().
- **Parameters**: The list of parameters you want to pass on the block, just like you do when declaring methods. However, keep in mind an important notice: When you pass no parameters to block, then the void keyword should be always set. The arguments list must also be enclosed in parentheses.

Example-

```
int (^firstBlock)(NSString *param1, int param2);


void (^showName)(NSString *myName);

NSDate *(^whatDayIsIt)(void);

void (^allVoid)(void);

NSString *(^composeName)(NSString *firstName, NSString *lastName);
```

Q25. What happens when you add your just created object to a mutable array & then release your object?

Releasing the array has the same effect as removing all the items from the array. That is, the array no longer claims ownership of them. If anyone else has retained the objects, they'll continue to exist. If not, they'll be deallocated.

This is just the normal set of memory rules in effect. If you retain an object, you must release it. Conversely, other bits of code (like an array) are allowed to retain and release that object when they want to own it, too, but if everyone follows the rules, nobody gets surprised.

Q26. What happens if we send any message to an object which is released? Which is abandoned?

Q27. What are the benefits of collection views?

- Allows customizable layout that can be very complex(example: different sizing of each cell)
- Supports multiple cells in one row
- Provides horizontal scroll direction, eliminates the need to create scroll view to achieve the same effect
- Supports custom animations for cells based on layout
- Can implement drag and drop functionality
- More options for headers and footers settings, it does not need to be anchored on the top or bottom
- Does not include blank cells

Q28. What is the relation between ivars and @property

private vs public
For a private/protected variable, use iVar; for a public variable, use property. If you want to use the benifit of property attributes for a private variable, like retain,

nonatomic etc., declare the property in the implementation file as a private property.
For an iVar, you can use @private, @protected and @public. But these attributes only influence the access of its subclass, and has nothing to do with the access ability of its instances. Go to here for reference.

usage
Directly use an iVar inside the class, for example, photographer. But use self. for a property, for example, self.photographer.

performance
iVar is faster than property, as property will call the getter or setter functions. When you declare a property, the compiler will add getter and setter functions for the property.

Q29. What is NSURLConnection class? Define its type and usage.

There are two ways of using NSURLConnection class. One is asynchronous and the other is synchronous. An asynchronous connection will create a new thread and performs its download process on the new thread. A synchronous connection will block the calling thread while downloading content and doing its communication. Many developers think that a synchronous connection blocks the main thread, which is not true. A synchronous connection will always block the thread from which it is fired. If you fire a synchronous connection from the main thread, the main thread will be blocked. But, if you fire a synchronous connection from a thread other than the main thread, it will be like an asynchronous connection and won't block your main thread. In fact, the only difference between a synchronous and an asynchronous connection is that at runtime, a thread will be created for the asynchronous connection while it won't do the same for a synchronous connection.
In order to create an asynchronous connection, we need to do the following:
1. Have our URL in an instance of NSString
2. Convert our string to an instance of NSURL
3. Place our URL in a URL Request of type NSURLRequest, or in the case of mutable URLs, in an instance of NSMutableURLRequest.
4. Create an instance of NSURLConnection and pass the URL request to it.

Q30. Difference between AppID and BundleID.

1. Bundle ID –

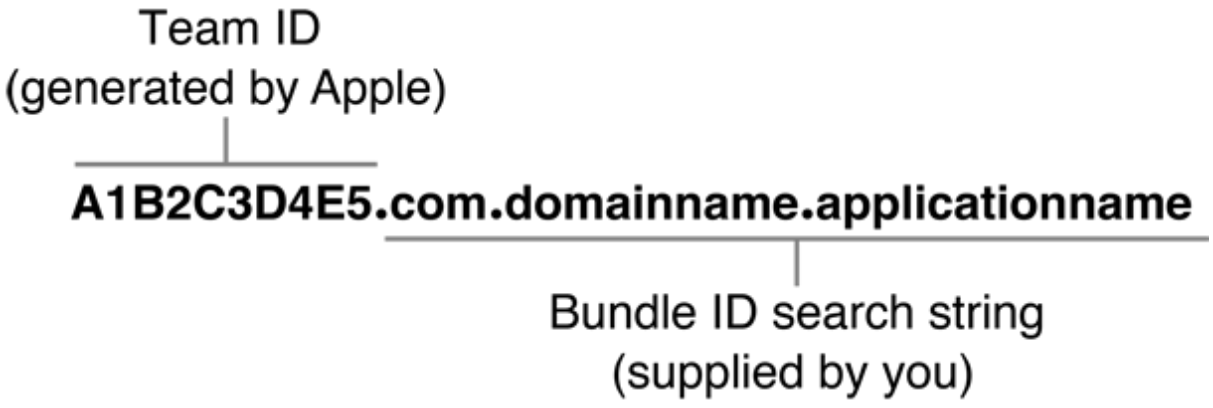
A bundle ID precisely identifies a single app.

The bundle ID string must be a uniform type identifier (UTI) that contains only alphanumeric characters (A-Z,a-z,0-9), hyphen (-), and period (.). The string should be in reverse-DNS format.

For example, if your organization's domain is abc.com and you create an app named Hello, you could assign the string com.abc.Hello as your app's bundle ID.

2. App ID -

An App ID is a two-part string used to identify one or more apps from a single development team. The string consists of a Team ID and a bundle ID search string, with a period (.) separating the two parts. The Team ID is supplied by Apple and is unique to a specific development team, while the bundle ID search string is supplied by you to match either the bundle ID of a single app or a set of bundle IDs for a group of your apps.

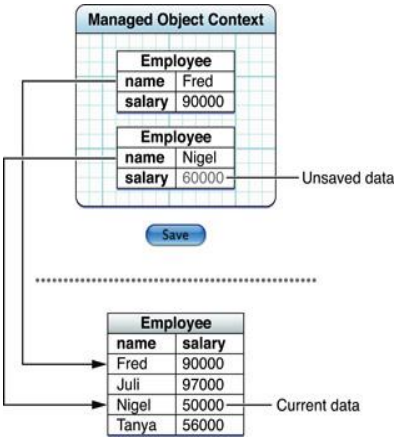


Q.31) Describe manage object context its function?

Answer-

A managed object context represents a single object space, or scratch pad, in a Core Data application. A managed object context is an instance of NSManagedObjectContext. Its primary responsibility is to manage a collection of managed objects. These managed objects represent an internally consistent view of one or more persistent stores. The context is a powerful object with a central role in the life-cycle of managed objects, with responsibilities from life-cycle management (including faulting) to validation, inverse relationship handling, and undo/redo.

the context is the central object in the Core Data stack. It's the object we use to create and fetch managed objects, and to manage undo and redo operations. Within a given context, there is at most one managed object to represent any given record in a persistent store.



A context is connected to a parent object store. This is usually a persistent store coordinator, but may be another managed object context. When you fetch objects, the context asks its parent object store to return those objects that match the fetch request. Changes that you make to managed objects are not committed to the parent store until you save the context.

Q.32) What is unnamed category?

Answer-

A named category "@interface Foo(MyCategory)" is generally used to:
– Extend an existing class by adding functionality.
– Declare a set of methods that might or might not be implemented by a delegate.

Unnamed Categories has fallen out of favor now that @protocol has been extended to support @optional methods.
A class extension "@interface Foo()" is designed to allow you to declare additional private API — SPI or System Programming Interface — that is used to implement the class innards. This typically appears at the top of the .m file. Any methods / properties declared in the class extension must be implemented in the @implementation, just like the methods/properties found in the public @interface.

Class extensions can also be used to redeclare a publicly readonly @property as readwrite prior to @synthesize'ing the accessors.

Example:
Foo.h
@interface Foo:NSObject
@property(readonly, copy) NSString *bar;
-(void) publicSaucing;
@end
Foo.m
@interface Foo()
@property(readwrite, copy) NSString *bar;
– (void) superSecretInternalSaucing;
@end
@implementation Foo
@synthesize bar;
…. must implement the two methods or compiler will warn ….
@end