Q-1 What is Parsing? Explain XML parsing and JSON parsing with example. OR Explain JSON parsing with example.

- Parsing defined as separation. To separate the sentence into grammatical meaning or words, phrase, numbers.

JSON parsing

- If your app communicates with a web application, information returned from the server is often formatted as JSON.
- You can use the Foundation framework's JSONSerialization class to convert JSON into Swift data types like Dictionary, Array, String, Number, and Bool.

Extracting Values from JSON

- The JSONSerialization class method jsonObject(with:options:) returns a value of type Any and throws an error if the data couldn't be parsed.

```
import Foundation
let data: Data // received from a network request, for example
let json = try? JSONSerialization.jsonObject(with: data, options: [])
```

- Although valid JSON may contain only a single value, a response from a web application typically encodes an object or array as the top-level object.
- You can use optional binding and the as? type cast operator in an if or guard statement to extract a value of known type as a constant.
- To get a Dictionary value from a JSON object type, conditionally cast it as [String: Any]. To get an Array value from a JSON array type, conditionally cast it as [Any] (or an array with a more specific element type, like [String]).
- You can extract a dictionary value by key or an array value by index using type cast optional binding with subscript accessors or pattern matching with enumeration.

```
// Example JSON with object root:
/*
    {
            "someKey": 42.0,
            "anotherKey": {
                    "someNestedKey": true
            }
    }
*/
if let dictionary = jsonWithObjectRoot as? [String: Any] {
    if let number = dictionary["someKey"] as? Double {
            // access individual value in dictionary
    }

    for (key, value) in dictionary {
            // access all key / value pairs in dictionary
    }
```

```
        if let nestedDictionary = dictionary["anotherKey"] as? [String: Any] {
                // access nested dictionary values by key
        }
    }
    // Example JSON with array root:
    /*
        [
                "hello", 3, true
        ]
    */
    if let array = jsonWithArrayRoot as? [Any] {
        if let firstObject = array.first {
                // access individual object in array
        }

        for object in array {
                // access all objects in array
        }

        for case let string as String in array {
                // access only string values in array
        }
    }
```

- Swift's built-in language features make it easy to safely extract and work with JSON data decoded with Foundation APIs — without the need for an external library or framework.

Creating Model Objects from Values Extracted from JSON

- Since most Swift apps follow the Model-View-Controller design pattern, it is often useful to convert JSON data to objects that are specific to your app's domain in a model definition.
- For example, when writing an app that provides search results for local restaurants, you might implement a Restaurant model with an initializer that accepts a JSON object and a type method that makes an HTTP request to a server's /search endpoint and then asynchronously returns an array of Restaurant objects.
- Consider the following Restaurant model:

```
    import Foundation

    struct Restaurant {
        enum Meal: String {
                case breakfast, lunch, dinner
        }

        let name: String
        let location: (latitude: Double, longitude: Double)
        let meals: Set<Meal>
    }
```

- A Restaurant has a name of type String, a location expressed as a coordinate pair, and a Set of meals containing values of a nested Meal enumeration.
- Here’s an example of how a single restaurant may be represented in a server response:

```
{
    "name": "Caffè Macs",
    "coordinates": {
            "lat": 37.330576,
            "lng": -122.029739
    },
    "meals": ["breakfast", "lunch", "dinner"]
}
```

Writing an Optional JSON Initializer

- To convert from a JSON representation to a Restaurant object, write an initializer that takes an Any argument that extracts and transforms data from the JSON representation into properties.

```
extension Restaurant {
    init?(json: [String: Any]) {
            guard let name = json["name"] as? String,
                    let coordinatesJSON = json["coordinates"] as? [String: Double],
                    let latitude = coordinatesJSON["lat"],
                    let longitude = coordinatesJSON["lng"],
                    let mealsJSON = json["meals"] as? [String]
            else {
                    return nil
            }

            var meals: Set<Meal> = []
            for string in mealsJSON {
                    guard let meal = Meal(rawValue: string) else {
                            return nil
                    }

                    meals.insert(meal)
            }

            self.name = name
            self.coordinates = (latitude, longitude)
            self.meals = meals
```

```
        }
    }
```

- If your app communicates with one or more web services that do not return a single, consistent representation of a model object, consider implementing several initializers to handle each of the possible representations.
- In the example above, each of the values are extracted into constants from the passed JSON dictionary using optional binding and the as? type casting operator.
- For the name property, the extracted name value is simply assigned as-is.
- For the coordinate property, the extracted latitude and longitude values are combined into a tuple before assignment.
- For the meals property, the extracted string values are iterated over to construct a Set of Meal enumeration values.

XML Parsing

- Generally, when you parse an XML document most of the processing involves elements and things related to elements, such as attributes and textual content.
- Elements hold most of the information in an XML document.
- To parse XML Data declare following variable in your class.

```
var parser = NSXMLParser()
var posts = NSMutableArray()
var elements = NSMutableDictionary()
var element = NSString()
var title1 = NSMutableString()
var date = NSMutableString()
```

- Here parser is the object, which is used to download and parse the xml file.
- Post is object of mutable array used to store feed data.
- Element is a mutabledictionary which contains the data of feed like title and date separately. Title1 and date used to store string data of feed.
- Now from viewDidLoad method calling beginParsing method in that doing initialization of parser object, set NSXMLParserDelegate and then start XML parsing.

```
func beginParsing()
 {
     posts = []
     parser = NSXMLParser(contentsOfURL:(NSURL(string: "http://images.
apple.com/main/rss/hotnews/hotnews.rss"))!)!
     parser.delegate = self
     parser.parse()
     tbData!.reloadData()
   }
```

- To implement the parser delegate method have to set delegate in class as shown below. It informs the compiler that ViewController class implements the NSXMLParserDelegate.

```
class ViewController: UIViewController, NSXMLParserDelegate
```

- During parsing, when parser finds any new element it calls the below delegate method. In this method allocate variable when parser find the item element.

```
func parser(parser: NSXMLParser, didStartElement elementName: String,
namespaceURI: String?, qualifiedName qName: String?, attributes
attributeDict: [String : String])
  {
     element = elementName
     if (elementName as NSString).isEqualToString("item")
     {
        elements = NSMutableDictionary()
        elements = [:]
        title1 = NSMutableString()
        title1 = ""
        date = NSMutableString()
        date = ""
     }
  }
```

- After that when it finds new character it calls the below delegate method. In this method append all character in mutable string for particular element.

```
func parser(parser: NSXMLParser!, foundCharacters string: String!)
  {
     if element.isEqualToString("title") {
        title1.appendString(string)
     } else if element.isEqualToString("pubDate") {
        date.appendString(string)
     }
  }
```

- When parser finds the end of element it calls the below delegate method. In that just store the feed data in dictionary and then add that dictionary in array.

```
func parser(parser: NSXMLParser!, didEndElement elementName: String!,
namespaceURI: String!, qualifiedName qName: String!)
  {
     if (elementName as NSString).isEqualToString("item") {
        if !title1.isEqual(nil) {
           elements.setObject(title1, forKey: "title")
        }
        if !date.isEqual(nil) {
           elements.setObject(date, forKey: "date")
        }

        posts.addObject(elements)
     }
  }
```

- After xml parsing display data from array in tableview.

Q-2 Explain COCOA Touch classes. Also give the differences between COCOA and COCOA Touch classes.

- Cocoa Touch is a UI framework for building software programs to run on iOS (for the iPhone, iPod Touch, and iPad), watchOS for the Apple Watch, and tvOS for the fourth-generation Apple TV, from Apple Inc.
- Some of the main features and technologies of Cocoa Touch are:
  - App Extension
  - Handoff
  - Document Picker
  - AirDrop
  - TextKit
  - UIKit Dynamics
  - Multitasking
  - Auto Layout
  - Storyboards
  - UI State Preservation
  - Apple Push Notification Service
  - Local Notifications
  - Gesture Recognisers
  - Standard System View Controllers
- Cocoa Touch provides the key frameworks for developing applications on devices running iOS. Some of these key frameworks are:
  - Foundation Kit Framework
  - UIKit Framework (based on Application Kit)
  - GameKit Framework
  - iAd Framework
  - MapKit Framework
  - Address Book UI Framework
  - EventKit UI Framework
  - Message UI Framework
  - Notification Center Framework
  - PushKit Framework
  - Twitter Framework
- Cocoa and Cocoa Touch are both widely popular for app development. The prime differences between these two are as follows:
  1. **iOS vs Mac development** – This is probably the most important difference between Cocoa and Cocoa Touch, from the usability perspective. The former is used for coding for Macs (Mac OS X), while the latter is used exclusively for iOS app development.
  2. **Underlying combination of frameworks** – Both Cocoa and Cocoa Touch make use of the Foundation framework. The application framework for Mac systems makes use of AppKit, which is not present in Cocoa Touch. Instead, Touch is a combination of UIKit and Foundation.
  3. **API differences** – All the classes used in Cocoa have the NS prefix (i.e., NSTextField and NSWindow). Classes in Cocoa Touch, on the other hand, are prefixed by UI

(i.e., UITextField and UIWindow).

4. **Better MVC patterns in Cocoa Touch –** The universality of the Model View Controller (MVC) is the only field where we can go for a direct comparison of the two frameworks. Cocoa Touch comes out on top in this regard. The iPhone SDK has a reinforced MVC system, which performs better than the default MVC in Cocoa. Of course, Cocoa has multiple alternative design patterns – in addition to MVC.
5. **Absence of certain classes** – There are certain classes in the Mac OS X development framework, that are not present in its iOS counterpart. For instance, Cocoa has NSHost, but no similar classes are there in Cocoa Touch.
6. **Difference in the system support** – There are certain unique features of Cocoa and Cocoa Touch, which can help a new app developer distinguish between the two. In particular, there are differences in the app lifecycle and sandboxing options in the two framework systems. Also, the total Memory Footprint in Cocoa and Cocoa Touch are significantly different.
7. **Device support** – Apple had released the Cocoa API for desktop app development only, following the days of Rhapsody and Yellow Box. The framework can be used for all types of Mac desktop app development. On the other hand, Cocoa Touch, with its additional animation features and gesture controls, is optimized for creating apps for iPhone, iPod Touch, iPad and even Apple TV. Over the next few months, Cocoa Touch is likely to become popular for making apps for Apple Watch as well.

Q-3 What is Framework? Explain COCOA framework in detail. OR What is framework? Explain COCOA and MVC framework. OR Explain MVC framework by giving proper example

- A framework is a bundle (a structured directory) that contains a dynamic shared library along with associated resources, such as nib files, image files, and header files.
- When you develop an application, your project links to one or more frameworks.
- For example, iPhone application projects link by default to the Foundation, UIKit, and Core Graphics frameworks.

COCOA Framework

- Cocoa is Apple's native object-oriented application programming interface (API) for their operating system macOS.
- Cocoa consists of the Foundation Kit, Application Kit, and Core Data frameworks, as included by the Cocoa.h header file, and the libraries and frameworks included by those, such as the C standard library and the Objective-C runtime.
- Cocoa applications are typically developed using the development tools provided by Apple, specifically Xcode and Interface Builder, using the languages Objective-C or Swift.

- However, the Cocoa programming environment can be accessed using other tools, such as Clozure CL, LispWorks, Object Pascal, Python, Perl, Ruby, and AppleScript with the aid of bridge mechanisms such as PasCocoa, PyObjC, CamelBones, RubyCocoa, and a D/Objective-C Bridge.
- One feature of the Cocoa environment is its facility for managing dynamically allocated memory.
- Cocoa's NSObject class, from which most classes, both vendor and user, are derived, implements a reference counting scheme for memory management.
- Objects that derive from the NSObject root class respond to retain and a release message, and keep a retain count.
- Cocoa consists of three Objective-C object libraries called frameworks.
- Frameworks are functionally similar to shared libraries, a compiled object that can be dynamically loaded into a program's address space at runtime, but frameworks add associated resources, header files, and documentation.
- Cocoa consists of three frameworks:

1. **Foundation Kit (Foundation):** It was developed as part of the OpenStep work, and subsequently became the basis for OpenStep's AppKit when that system was released in 1994. On macOS, Foundation is based on Core Foundation. Foundation is a generic object-oriented library providing string and value manipulation, containers and iteration, distributed computing, event loops (run loops), and other functions that are not directly tied to the graphical user interface. The "NS" prefix, used for all classes and constants in the framework.
2. **Application Kit (AppKit):** It is directly descended from the original NeXTSTEP Application Kit. It contains code programs can use to create and interact with graphical user interfaces. AppKit is built on top of Foundation, and uses the same NS prefix.
3. **Core Data:** It is the object persistence framework included with Foundation and Cocoa and found in Cocoa.h

- A key part of the Cocoa architecture is its comprehensive views model.
- This is organized along conventional lines for an application framework, but is based on the Portable Document Format (PDF) drawing model provided by Quartz.
- This allows creating custom drawing content using PostScript-like drawing commands, which also allows automatic printer support and so forth.
- Since the Cocoa framework manages all the clipping, scrolling, scaling and other chores of drawing graphics, the programmer is freed from implementing basic infrastructure and can concentrate on the unique aspects of an application's content.

Model-View-Controller (MVC) Framework

- The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller.
- The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other.
- Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries.

- The collection of objects of a certain MVC type in an application is sometimes referred to as a layer—for example, model layer.
- MVC is central to a good design for a Cocoa application.
- Many objects in these applications tend to be more reusable, and their interfaces tend to be better defined.
- Applications having an MVC design are also more easily extensible than other applications.
- Moreover, many Cocoa technologies and architectures are based on MVC and require that your custom objects play one of the MVC roles.

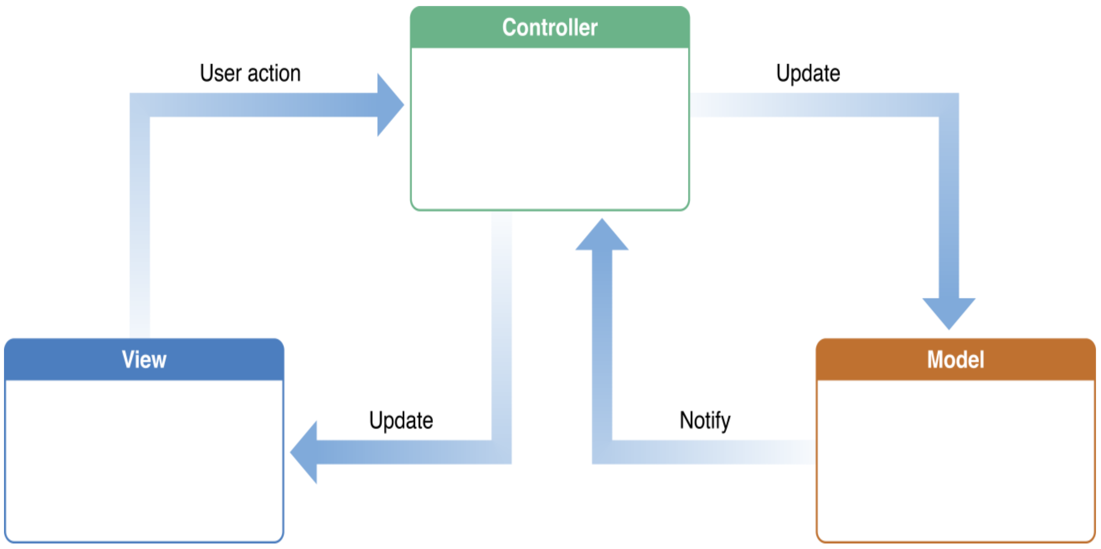

Fig. 1: MVC Framework

Model object

- Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data.
- For example, a model object might represent a character in a game or a contact in an address book.
- A model object can have to-one and to-many relationships with other model objects, and so sometimes the model layer of an application effectively is one or more object graphs.
- Much of the data that is part of the persistent state of the application (whether that persistent state is stored in files or databases) should reside in the model objects after the data is loaded into the application.
- Because model objects represent knowledge and expertise related to a specific problem domain, they can be reused in similar problem domains.
- Ideally, a model object should have no explicit connection to the view objects that present its data and allow users to edit that data—it should not be concerned with user-interface and presentation issues.
- **Communication:** User actions in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object. When a model object changes (for example, new data is received over a network connection), it notifies a controller object, which updates the appropriate

view objects.

View Objects

- A view object is an object in an application that users can see.
- A view object knows how to draw itself and can respond to user actions.
- A major purpose of view objects is to display data from the application's model objects and to enable the editing of that data.
- Despite this, view objects are typically decoupled from model objects in an MVC application.
- Because you typically reuse and reconfigure them, view objects provide consistency between applications.
- Both the UIKit and AppKit frameworks provide collections of view classes, and Interface Builder offers dozens of view objects in its Library.
- **Communication:** View objects learn about changes in model data through the application's controller objects and communicate user-initiated changes—for example, text entered in a text field—through controller objects to an application's model objects.

Controller Objects

- A controller object acts as an intermediary between one or more of an application's view objects and one or more of its model objects.
- Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa.
- Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.
- **Communication:** A controller object interprets user actions made in view objects and communicates new or changed data to the model layer. When model objects change, a controller object communicates that new model data to the view objects so that they can display it.

Example:

- Imagine you have a UIViewController subclass that wants to know the list of WWDC attendees this year.
- To achieve this, it makes use of a controller class. Since Apple's been preaching that we should always start with a protocol, you'll do that here:

```
enum UIState {
   case Loading
   case Success([Attendee])
   case Failure(Error)
}

protocol WWDCAttendesDelegate: class {
   var state: UIState { get set}
}
```

- The idea is that you'll initially set state to Loading, then update it when the list of WWDC attendees is successfully loaded (or fails).
- Since you don't want the UIViewController to handle the response, it will use a

separate object (WWDCAttendeesUIController) that will implement WWDCAttendesDelegate.

- This separation, allows you to easily test WWDCAttendeesUIController independently.
- The next step is to create an abstraction for the controller, so you can inject it into your UIViewController:

```
protocol WWDCAttendeesHandler: class {

   var delegate: WWDCAttendesDelegate? { get set }
   func fetchAttendees()
}
```

- From the point of view of the UIViewController subclass, the implementation would look like this:

```
init(attendeesHandler: WWDCAttendeesHandler) {

   self.attendeesHandler = attendeesHandler
   super.init(nibName: nil, bundle: nil)
}

override func viewDidLoad() {
   super.viewDidLoad()

   atteendeesUIController = WWDCAttendeesUIController(view: view,
tableView: tableView)
   attendeesHandler.delegate = atteendeesUIController

   attendeesHandler.fetchAttendees()
}
```

- This approach will put the fetching action on the UIViewController side, but leave the response handling to the WWDCAttendeesUIController:

```
extension WWDCAttendeesUIController: WWDCAttendesDelegate {

   func update(newState: UIState) {

      switch(state, newState) {

      case (.Loading, .Loading): loadingToLoading()
      case (.Loading, .Success(let attendees)): loadingToSuccess(attendees)

      default: fatalError("Not yet implemented \(state) to \(newState)")
      }
   }

   func loadingToLoading() {
      view.addSubview(loadingView)
```

```
        loadingView.frame = CGRect(origin: .zero, size: view.frame.size)
    }

    func loadingToSuccess(attendees: [Attendee]) {
        loadingView.removeFromSuperview()
        tableViewDataSource.dataSource                                        =
attendees.map(AttendeeCellController.init)
    }
}
```

- You can see the WWDCAttendeesUIController as the UI brain, while the WWDCAttendeesController as the business logic brain.
- Making the controller injectable — so the owner of your UIViewController would provide the controller.
- This has two main benefits:
    - It's easily testable. You can simply pass any object that complies with the FetchNumberOfTickets protocol.
    - The layers are cleanly decoupled. This helps in defining responsibilities, which leads to an overall healthier code base.

Q-4 Enlist all features of latest iOS. OR Overview of features of latest iOS. OR Explain the features of iOS 7 and iOS 8.

Features of iOS 7

- **Design:**
    - iOS 7 introduced a complete visual overhaul of the user interface.
    - With "sharper, flatter icons, slimmer fonts, a new slide-to-unlock function, and a new control panel that slides up from the bottom of the screen for frequently accessed settings", the operating system also significantly redesigned the standard pre-installed apps from Apple.
- **Airdrop:**
    - iOS 7 introduced AirDrop, a wireless sharing technology accessible from the share icon, which lets users send files from their local device to other, nearby iOS devices.
    - It can be enabled in the Control Center, with options for controlling its discovery by other devices, including Off, Contacts Only, and Everyone.
- **Control center:**
    - iOS 7 introduced the Control Center, a menu that users can access by swiping up from the bottom of the screen, providing access to frequently used settings such as Airplane Mode, Wi-Fi, Bluetooth, Do Not Disturb Mode, and Rotation Lock.
    - There is a slider for adjusting screen brightness, controls for music playback, along with a volume slider.
    - A Flashlight toggle is also available, as is an icon for quickly accessing the

camera.

- **Notification center:**
  - iOS 7 overhauled the Notification Center, with both visual and functional changes.
  - It has three tabs; "Today" (information on what's coming during the day), "All", and "Missed".
  - Notifications are now visible on the phone's lock screen, and notifications are synchronized across devices, so users don't need to dismiss the same notification multiple times on different devices.
- **Multitasking:**
  - iOS 7 expanded the multitasking screen.
  - Users double-tapped the home button and were taken out of the current app for a view of full-screen thumbnails of recently accessed apps.
- **CarPlay:**
  - CarPlay (formerly iOS in the Car), released as part of iOS 7.1, integrates with selected car models to offer phone interaction, including music controls and Maps navigation, through the car's display.
  - Users can also talk to Siri to send messages hands-free.
  - It supports all Lightning-equipped iPhones with iOS 7.1 or later.
- **Siri:**
  - Siri features a new design where the assistant fades in with the background app being blurred.
  - While speaking, visual vocal waves appear, that form into a spinning circle as the device communicates with Apple's servers.
  - Siri can now control a limited set of device settings, including Bluetooth.
  - The assistant can alternatively send the user directly to the settings menu if their query cannot be controlled by voice, or otherwise, displays an error if the request can't be processed.
  - For searching the Internet, Bing became the default search engine, and Siri's web resources were expanded to include Wikipedia.
  - Siri was also more deeply connected into Twitter, offering full tweet search.
  - iOS 7 also allowed the user to change Siri's gender, with new options for male and female voices.
- **Other:**
  - Spotlight search is accessed by holding and dragging down the home screen.
  - iOS 7 came with new wallpapers that included a "Parallax" effect; the icons appear to be moving as the user moves the phone around, producing an "illusion that the icons are floating above the background wallpaper".
  - In Settings, users have the option to make custom vibrations for certain types of notifications, turn off the parallax wallpaper animation effect using a Reduce Motion setting, and block callers, with the block applying across multiple devices and apps, including Phone, Messages, and FaceTime.

Features of iOS 8

- **Continuity:**
  - iOS 8 introduced Continuity, a cross-platform (Mac, iPhone, and iPad) system

  that enables communication between devices in different product categories.
  - Continuity enables phone call functionality for the iPad and Mac, in which calls are routed through the iPhone over to a secondary device.
  - The secondary device then serves as a speaker phone. This also brings SMS support to the iPad and Mac, an extension of the iMessage feature in previous versions.
  - Continuity adds a feature called "Handoff", that lets users start a task on one device and continue on another, such as composing an e-mail on the iPhone and then continuing it on the iPad before sending it on the Mac.
  - In order to support Handoff and Continuity, Macs needed to have the OS X Yosemite operating system as well as support for Bluetooth low energy
- **Spotlight:**
  - iOS 8 introduced Spotlight Suggestions, a new search feature that integrates with many websites and services to show more detailed search results, including snippets of Wikipedia articles, local news, quick access to apps installed on the device, iTunes content, movie show times, nearby places, and info from various websites.
  - Spotlight Suggestions are available on the iOS home screen as well as in the Safari web browser search bar.
- **Notifications:**
  - The drop-down Notification Center has now been redesigned to allow widget functionality.
  - Third-party apps can add widget support to their apps that let users see information in the Notification Center without having to open each respective app.
  - Users can add, rearrange, or remove any widgets, at any time.
  - Examples of widgets include a Weather app showing current weather, and a Calendar app showing upcoming events.
  - Notifications are now actionable, allowing users to reply to a message while it appears as a quick drop-down, or act on a notification through the Notification Center.
- **Keyboard:**
  - iOS 8 includes a new predictive typing feature called QuickType, which displays word predictions above the keyboard as the user types.
  - Apple now allows third-party developers to make keyboard apps that users can replace the default iOS keyboard with.
  - For added privacy, Apple added a settings toggle called "Allow Full Access", that optionally enables the keyboard to act outside its app sandbox, such as synchronizing keyboard data to the cloud, third-party keyboards are not allowed to use Siri for voice dictation, and some secure text fields do not allow input.
- **Family Sharing:**
  - iOS 8 introduced Family Sharing, which allows up to 6 people to register unique iTunes accounts that are then linked together, with one parent becoming the administrator, controlling the overall experience.
  - Purchases made on one account can be shared with the other family

members, but purchases made by kids under 13 years of age require parental approval. Purchases made by adults will not be visible for the kids at all.
    - Family Sharing also extends into apps; a Shared album is automatically generated in the Photos app of each family member, allowing everyone to add photos, videos, and comments to a shared place.
    - An Ask to Buy feature allows anyone to request the purchase of items in the App Store, iTunes Store, and iBooks Store, as well as in-app purchases and iCloud storage, with the administrator having the option to either approve or deny the purchase.
- **Multitasking:**
    - The multitasking screen shows a list of recently called and favorite contacts. The feature can be turned off in Settings.
- **Other:**
    - iOS 8 includes an additional data roaming option in Settings for European users, allowing greater control over data usage abroad.
    - The Siri personal voice assistant now has integrated Shazam support.
    - Asking Siri "What song is this?" will identify what song is playing.
    - Wi-Fi calling has been added to allow mobile phone calls over Wi-Fi.
    - Mobile operator carriers can then enable the Voice-over-Wi-Fi functionality in their services.

Q-5 Explain alert view and table view with suitable example. OR What is view? Explain alert views, table views, picker, date and time. OR Give code to create a simple Table View application in iOS. OR Create a Simple Table View Application.

- The View defines a rectangular area on the screen and the interfaces for managing the content in that area.
- At runtime, a view object handles the rendering of any content in its area and also handles any interactions with that content.

Alert View

- Alert views display an informative alert message to the user.
- Alert views interrupts the user and requires them to stop what they're doing to choose an action or dismiss the alert.

Example:

- Open Xcode and create a new Single View Application.
- Go to the Storyboard.
- Drag a button to the main view and give it a title of "Show Alert".
- Select the Assistant Editor and open ViewController.swift. Ctrl and drag from the button to the class and create the following Action:

```
@IBAction func buttonTapped(sender: AnyObject) {

}
```

- Note in Swift the sender object is of type AnyObject, this is the equivalent of the id object in Objective-C. Next, implement the buttonTapped function.

```
@IBAction func buttonTapped(sender: AnyObject) {
  let alertController = UIAlertController(title: "Alert", message:
   "Hello, world!", preferredStyle: UIAlertControllerStyle.Alert)
 alertController.addAction(UIAlertAction(title: "Dismiss", style:
UIAlertActionStyle.Default,handler: nil))
  self.presentViewController(alertController, animated: true, completion: nil)
}
```

- The alerController constant is assigned an UIAlertControllerObject. Since we will show an Alert View the preferredStyle is set to Alert.
- The alertController is then presented with a call to presentViewController.
- Build a Run the project and press the Show Alert button to display the alert view.
- It will show you the alert with title “Alert” and message “Hello, world!”

Table View

- It is used for displaying a vertically scrollable view which consists of a number of cells (generally reusable cells).
- It has special features like headers, footers, rows, and section.
- To create a table view, several entities in an app must interact: the view controller, the table view itself, and the table view’s data source and delegate.
- The view controller, data source, and delegate are usually the same object.
- The view controller starts the calling sequence.
- The view controller creates a UITableView instance in a certain frame and style.
- It can do this either programmatically or in a storyboard.
- The frame is usually set to the screen frame, minus the height of the status bar or, in a navigation-based app, to the screen frame minus the heights of the status bar and the navigation bar.
- The view controller may also set global properties of the table view at this point, such as its autoresizing behavior or a global row height.
- The view controller sets the data source and delegate of the table view and sends a reloadData message to it.
- The data source must adopt the UITableViewDataSource protocol, and the delegate must adopt the UITableViewDelegate protocol.
- The data source receives a numberOfSectionsInTableView: message from the UITableView object and returns the number of sections in the table view.
- Although this is an optional protocol method, the data source must implement it if the table view has more than one section.
- For each section, the data source receives a tableView:numberOfRowsInSection: message and responds by returning the number of rows for the section.
- The data source receives a tableView:cellForRowAtIndexPath: message for each

visible row in the table view.

- It responds by configuring and returning a UITableViewCell object for each row.
- The UITableView object uses this cell to draw the row.

Example:

- Open Xcode and create a new Single View Application.
- Go to the Storyboard and drag a Table View on top of the main view.
- Ctrl and drag from the Table View to the View Controller in the Document Outline and select the dataSource outlet.
- Repeat this for the delegate outlet.
- In Swift there are no separate header and implementation files, the whole View Controller class is defined in the ViewController.swift file.
- Go to this file and right after the class ViewController: UIViewController { line add a constant containing some data for the table rows.

```
let tableData = ["One","Two",Three"]
```

- Swift has inferred the constant as an Array of Strings.
- Next, set the number of rows in the tableView:numberOfRowsInSection function.
- Add this function before the closing bracket of the class.

```
func tableView(tableView: UITableView!, numberOfRowsInSection section: Int)
-> Int
{
   return self.tableData.count;
}
```

- Our Table View will have 3 rows according to the count method of the Array class.
- Next, implement the tableView:cellForRowAtIndexPath function

```
func tableView(tableView: UITableView!,
       cellForRowAtIndexPath indexPath: NSIndexPath!) -> UITableViewCell!
{
   let cell:UITableViewCell = UITableViewCell(style: UITableViewCellStyle.
Default, reuseIdentifier:"cell")
   cell.textLabel?.text = tableData[indexPath.row]
   return cell
}
```

- Each Row will have a default tableview-style and will have the values of our tableData array.
- In Swift the intializer method names are truncated, so the initWithStyle: reuse Identifier method from Objective-C is named style:reuseIdentifier in Swift.
- Build and Run the project.
- It will show Table View with the rows filled.
- The table view with three rows namely: one, two, three will generated.

Picker

- The picker view is a slot-machine view to show one or more sets of values.
- Users select values by rotating the wheels so that the desired row of values aligns with a selection indicator.
- The user interface provided by a picker view consists of components and rows.
- A component is a wheel, which has a series of rows at indexed locations on the wheel.

Example:

- Open Xcode and create a new Single View Application.
- Go to the Storyboard, drag a Picker View from the Object Library to the top of the View Controller inside the Storyboard.
- Select the Picker View and set the proper location where you want to put the picker.
- The Picker View must conform to the UIPickerViewDataSource and UIPickerViewDelegate protocol.
- Ctrl-click the Picker View and drag from the dataSource Outlet to the View Controller in the Document Outline.
- Repeat this step for the delegate Outlet.
- We must provide the Picker View with values.
- Add the following array in the ViewController class in the ViewController.swift class:
```
var colors = ["Red","Yellow","Green","Blue"]
```
- The colors array will be the data source for our Picker View.
- The UIViewDataSource protocol requires delegate methods to define the number of components and rows of a picker.
- Implement these methods:
```
func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
  return 1
}

func pickerView(pickerView: UIPickerView, numberOfRowsInComponent
component: Int) -> Int {
   return colors.count
}
```
- We define one component with the number of rows equal to the number of array items.
- Next, we assign the data in out array to the corresponding row.
```
func pickerView(pickerView: UIPickerView, titleForRow row: Int,
forComponent component: Int) -> String! {
   return colors[row]
}
```
- Build and Run the project, the different colors can be selected inside the Picker View.

Date and Time

- NSDate class can be used to work with date and time.
- NSDate objects represent a single point in time.

- NSDate is a class cluster, its single public superclass.
- NSDate declares the programmatic interface for specific and relative time values.
- The object you create using NSDate are referred to as date objects.
- NSDate is an abstract class that provides behavior for creating dates, comparing dates, representing dates, computing intervals, and similar functionality.

Q-6 What is Object Oriented Programming? Provide a list of major difference between Objective C and C++.

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.
- A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated.
- In OOP, computer programs are designed by making them out of objects that interact with one another.
- There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.
- Many of the most widely used programming languages (such as C++, Delphi, Java, Python etc.) are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming.
- Significant object-oriented languages include Java, C++, C#, Python, PHP, Ruby, Perl, Object Pascal, Objective-C, Swift, Scala, Common Lisp, and Smalltalk.

Difference between Objective C and C++

- C++ allows multiple inheritance, Objective-C doesn't.
- Unlike C++, Objective-C allows method parameters to be named and the method signature includes only the names and types of the parameters and return type. In comparison, a C++ member function signature contains the function name as well as just the types of the parameters/return (without their names).
- C++ uses bool, true and false, Objective-C uses BOOL, YES and NO.
- C++ uses void* and nullptr, Objective-C prefers id and nil.
- Objective-C uses a messaging paradigm where you can send "messages" to objects through methods/selectors. Objective-C will happily let you send a message to nil, unlike C++ which will crash if you try to call a member function of nullptr.
- Objective-C allows for dynamic dispatch, allowing the class responding to a message to be determined at runtime, unlike C++ where the object a method is invoked upon must be known at compile time.
- Objective-C allows autogeneration of accessors for member variables using "properties".
- Objective-C allows assigning to self, and allows class initializers (similar to

constructors) to return a completely different class if desired. Contrast to C++, where if you create a new instance of a class. It is guaranteed to be of the type you originally specified.

- Similarly, in Objective-C other classes may also dynamically alter a target class at runtime to intercept method calls.
- Objective-C lacks the namespace feature of C++.
- Objective-C lacks an equivalent to C++ references.
- Objective-C lacks templates, preferring (for example) to instead allow weak typing in containers.
- Objective-C doesn't allow implicit method overloading, but C++ does.
- Objective-C will happily allow a method and a variable to share the same name, unlike C++ which will typically have fits. I imagine this is something to do with Objective-C using selectors instead of function pointers, and thus method names not actually having a "value".
- Objective-C doesn't allow objects to be created on the stack - all objects must be allocated from the heap.

Q-7 Explain NSString, NSMutableString, NSArray and NSMutable Array in details with suitable example.

NSString:

- NSString is the Cocoa object version of a string.
- NSString and Swift String are bridged to one another, and you will often move between them without thinking, passing a Swift String to Cocoa where an NSString is expected, calling Cocoa NSString methods on a Swift String, and so forth.
- For example:

```
let s = "hello"
let s2 = s.capitalized
```

- In that code, s is a Swift String and s2 is a Swift String, but the capitalized property actually belongs to Cocoa.
- In the course of that code, a Swift String has been bridged to NSString and passed to Cocoa, which has processed it to get the capitalized string; the capitalized string is an NSString, but it has been bridged back to a Swift String.
- In some cases, Swift may fail to cross the bridge implicitly for you, and you will need to cross the bridge yourself by casting explicitly.
- For example, if s is a Swift string, you can’t call appendingPathExtension on it directly:

```
let s = "MyFile"
let s2 = s.appendingPathExtension("txt") // compile error
```

- You have to cast explicitly to NSString.
- Another important difference between a Swift String and a Cocoa NSString is that an NSString is immutable.

- This means that, with NSString, you can do things such as obtain a new string based on the first — as capitalized and substring(to:) do — but you can't change the string in place.
- To do that, you need another class, a subclass of NSString, NSMutableString.

NSMutableString:

- NSMutableString – Should be used when you are physically changing the value of an existing string, without completely discarding the old value (i.e. adding a character to the beginning or end, modifying a character in the middle etc).
- NSMutableString is a subclass of NSString. So any method which can take an NSString can also take an NSMutableString.

NSArray:

- NSArray is Objective-C's array object type.
- It is fundamentally similar to Swift Array, and they are bridged to one another; but NSArray elements must be objects (classes and class instances), and they don't have to be of a single type.
- An NSArray's length is its count, and an element can be obtained by index number using object(at:).
- The index of the first element, as with a Swift Array, is zero, so the index of the last element is count minus one.
- Instead of calling object(at:), you can use subscripting with an NSArray.
- You can seek an object within an array with index(of:) or indexOfObject- Identical(to:) If the object is not found in the array, the result is NSNotFound.
- Unlike a Swift Array, and like an Objective-C NSString, an NSArray is immutable.
- This doesn't mean you can't mutate any of the objects it contains; it means that once the NSArray is formed you can't remove an object from it, insert an object into it, or replace an object at a given index.
- To do those things while staying in the Objective-C world, you can derive a new array consisting of the original array plus or minus some objects, or use NSArray's subclass, NSMutableArray.

NSMutableArray:

- The NSMutableArray class declares the programmatic interface to objects that manage a modifiable array of objects.
- This class adds insertion and deletion operations to the basic array-handling behavior inherited from NSArray.

Q-8 What is Object Mutability? Explain why Mutable and Immutable Objects are Variants?

- Cocoa objects are either mutable or immutable.
- You cannot change the encapsulated values of immutable objects; once such an object is created, the value it represents remains the same throughout the object's life.
- But you can change the encapsulated value of a mutable object at any time.

Why Mutable and Immutable Object Variants?

- Objects by default are mutable.
- Most objects allow you to change their encapsulated data through setter accessor methods.
- For example, you can change the size, positioning, title, buffering behavior, and other characteristics of an NSWindow object.
- A well-designed model object—say, an object representing a customer record—requires setter methods to change its instance data.
- The Foundation framework adds some nuance to this picture by introducing classes that have mutable and immutable variants.
- The mutable subclasses are typically subclasses of their immutable superclass and have "Mutable" embedded in the class name.
- These classes include the following:
    - NSMutableArray
    - NSMutableDictionary
    - NSMutableSet
    - NSMutableIndexSet
    - NSMutableCharacterSet
    - NSMutableData
    - NSMutableString
    - NSMutableAttributedString
    - NSMutableURLRequest
- Consider a scenario where all objects are capable of being mutated.
- In your application you invoke a method and are handed back a reference to an object representing a string.
- You use this string in your user interface to identify a particular piece of data.
- Now another subsystem in your application gets its own reference to that same string and decides to mutate it.
- Suddenly your label has changed out from under you. Things can become even direr if, for instance, you get a reference to an array that you use to populate a table view.
- The user selects a row corresponding to an object in the array that has been removed by some code elsewhere in the program, and problems ensue.
- Immutability is a guarantee that an object won't unexpectedly change in value while you're using it.
- Objects that are good candidates for immutability are ones that encapsulate collections of discrete values or contain values that are stored in buffers.
- But not all such value objects necessarily benefit from having mutable versions.
- Objects that contain a single simple value, such as instances of NSNumber or NSDate, are not good candidates for mutability.
- When the represented value changes in these cases, it makes more sense to replace the old instance with a new instance.
- Performance is also a reason for immutable versions of objects representing things such as strings and dictionaries.
- Mutable objects for basic entities such as strings and dictionaries bring some overhead with them.

- Because they must dynamically manage a changeable backing store—allocating and deallocating chunks of memory as needed—mutable objects can be less efficient than their immutable counterparts.
- Although in theory immutability guarantees that an object's value is stable, in practice this guarantee isn't always assured.
- A method may choose to hand out a mutable object under the return type of its immutable variant; later, it may decide to mutate the object, possibly violating assumptions and choices the recipient has made based on the earlier value.
- The mutability of an object itself may change as it undergoes various transformations.
- For example, serializing a property list (using the NSPropertyListSerialization class) does not preserve the mutability aspect of objects, only their general kind—a dictionary, an array, and so on.
- Thus, when you deserialize this property list, the resulting objects might not be of the same class as the original objects.
- For instance, what was once an NSMutableDictionary object might now be an NSDictionary object.

Q-9 How to read PDF file in iPhone simulator? Explain with code.

- A PDFView object encapsulates the functionality of PDF Kit into a single widget that you can add to your application using Interface Builder.
- PDFView may be the only class you need to deal with for adding PDF functionality to your application.
- It lets you display PDF data and allows users to select content, navigate through a document, set zoom level, and copy textual content to the Pasteboard.
- PDFView also keeps track of page history.
- You can subclass PDFView to create a custom PDF viewer.
- You can also create a custom PDF viewer by using the PDFViewer by using the PDF Kit utility classes directly and not using PDFView at all.
- Creating a PDFView is very simple thing to do. The code for that:

```
import Quartz

let url = NSBundle.mainBundle().URLForResource(“myPDF”, withExtension: ”pdf”)
let pdf = PDFDocument(URL: url)
let view = PDFView(frame: CGRect(x:0, y:0, width:500, height:750))
view.setDocument(pdf)
```

- In above code, the URLForResource() function search for the pdf file with given name in first argument and second argument is extension of file means 'pdf' in our case.
- URLForResource() function return the path of the desired pdf and we have stored the path in the url variable.
- We have created a PDFView object with desired width and height and given name it to view.
- View, the object of PDFView, will show our desired pdf into view when it will called.

- You could convert an entire PDF to a single NSAttributedString. Code for this is as follows:

```
import Quartz

let url = NSBundle.mainBundle().URLForResource("myPDF", withExtension: "pdf")
let pdf = PDFDocument(URL: url)
let docStr = NSMutableAttributedString()
for i in 0 ..< pdf.pageCount()
{
        docStr.appendAttributedString(doc.pageAtIndex(i).attributedString())
}
```

- But this would take a good deal of time for a long PDF and if you are looking to simply display the PDF then PDFView is the class which can be very useful.

Q-10 Explain MAC OS architecture and its features. OR Explain the basic architecture and features of the MAC OS.

- The architecture of macOS describes the layers of the operating system that is the culmination of Apple Inc.'s decade-long search and development process to replace the classic Mac OS.
- MAC OS provides many benefits to the Macintosh user and developer communities.
- These benefits include improved reliability and performance, enhanced networking features, an object-based system programming interface, and increased support for industry standards.
- In creating OS X, Apple has completely re-engineered the Mac OS core operating system.
- Forming the foundation of OS X is the kernel. Figure 2 illustrates the OS X architecture.

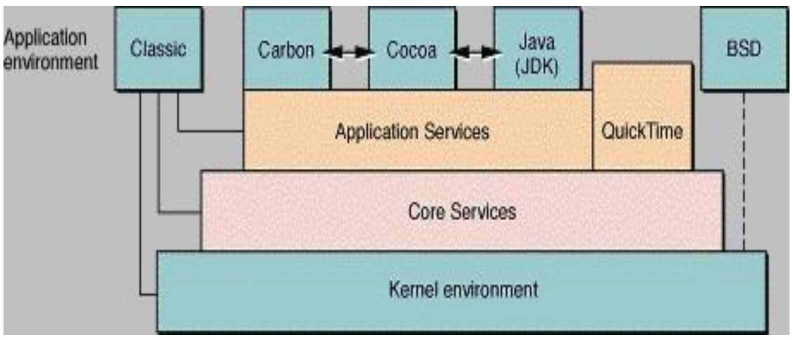

Fig. 2: MAC OS architecture

- The kernel provides many enhancements for OS X. These include preemption, memory protection, enhanced performance, improved networking facilities, support for both Macintosh and non-Macintosh file systems, object-oriented APIs, and more.
- Two of these features, preemption and memory protection, lead to a more robust environment.
- In Mac OS, applications cooperate to share processor time.

- Mac OS is a cooperative multitasking environment.
- The responsiveness of all processes is compromised if even a single application doesn't cooperate.
- On the other hand, real-time applications such as multimedia need to be assured of predictable, time-critical, behavior.
- The kernel provides enforcement of cooperation, scheduling processes to share time (preemption). This supports real-time behavior in applications that require it.
- In MAC OS, processes do not normally share memory.
- Instead, the kernel assigns each process its own address space, controlling access to these address spaces.
- This control ensures that no application can inadvertently access or modify another application's memory (protection).
- Size is not an issue; with the virtual memory system included in OS X, each application has access to its own 4 GB address space.
- All applications are said to run in user space, but this does not imply that they share memory.
- User space is simply a term for the combined address spaces of all user-level applications.
- The kernel itself has its own address space, called kernel space.
- In MAC OS, no application can directly modify the memory of the system software (the kernel).

Darwin:

- The MAC OS kernel is an Open Source project.
- The kernel, along with other core parts of OS are collectively referred to as Darwin.
- Darwin is a complete operating system based on many of the same technologies that underlie MAC OS.
- However, Darwin does not include Apple's proprietary graphics or applications layers, such as Quartz, QuickTime, Cocoa, Carbon, or OpenGL.
- Figure 3 shows the relationship between Darwin and OS X.

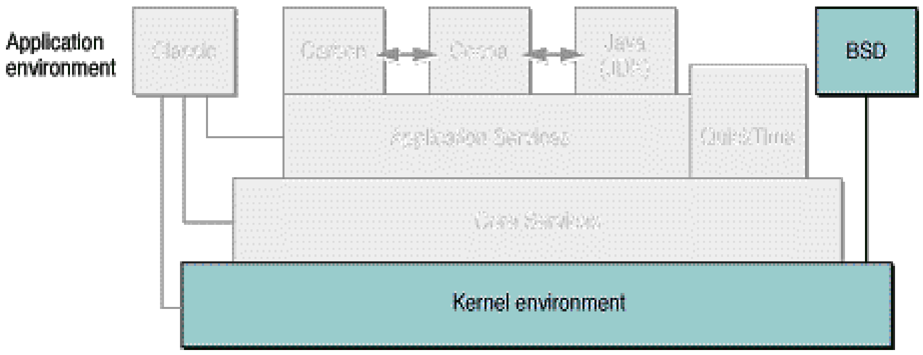

Fig. 3: Relationship between Darwin and MAC OS

- Both build upon the same kernel, but OS adds Core Services, Application Services and QuickTime, as well as the Classic, Carbon, Cocoa, and Java (JDK) application environments.

- Both Darwin and MAC OS include the BSD command-line application environment; however, use of environment is not required, and thus it is hidden from the user unless they choose to access it.
- Darwin technology is based on BSD, Mach 3.0, and Apple technologies.
- Best of all, Darwin technology is Open Source technology, which means that developers have full access to the source code.
- In effect, MAC OS third-party developers can be part of the Darwin core system software development team.
- Developers can also see how Apple is doing things in the core operating system and adopt code to use within their own products.
- Because the same software forms the core of both MAC OS and Darwin, developers can create low-level software that runs on both MAC OS and Darwin with few, if any, changes.
- The only difference is likely to be in the way the software interacts with the application environment.
- Darwin is based on proven technology from many sources. A large portion of this technology is derived from FreeBSD, a version of 4.4BSD that offers advanced networking, performance, security, and compatibility features.
- The core technologies have been chosen for several reasons.
- Mach provides a clean set of abstractions for dealing with memory management, interprocess (and interprocessor) communication (IPC), and other low-level operating-system functions.
- In today's rapidly changing hardware environment, this provides a useful layer of insulation between the operating system and the underlying hardware.
- BSD is a carefully engineered, mature operating system with many capabilities.
- In fact, most of today's commercial UNIX and UNIX-like operating systems contain a great deal of BSD code.
- BSD also provides a set of industry-standard APIs.
- New technologies, such as the I/O Kit and Network Kernel Extensions (NKEs), have been designed and engineered by Apple to take advantage of advanced capabilities, such as those provided by an object-oriented programming model.
- MAC OS combines these new technologies with time-tested industry standards to create an operating system that is stable, reliable, flexible, and extensible.

OS X kernel architecture

- The foundation layer of Darwin and OS X is composed of several architectural components, as shown in Figure 4. Taken together, these components form the kernel environment.

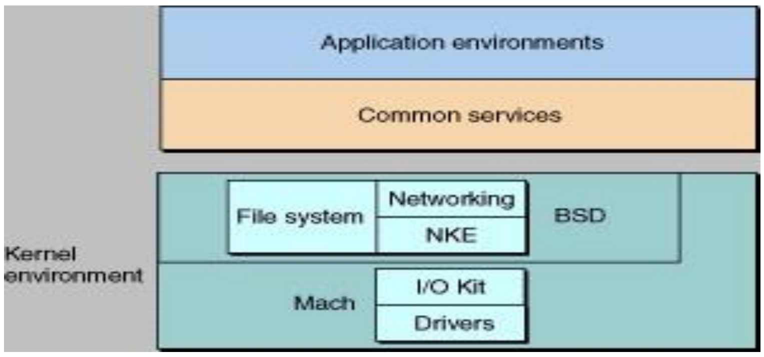

Fig. 4: kernel architecture

- In MAC OS, the kernel environment contains much more than the Mach kernel itself.
- The MAC OS kernel environment includes the Mach kernel, BSD, the I/O Kit, file systems, and networking components.
- Because MAC OS contains three basic components (Mach, BSD, and the I/O Kit), there are also frequently as many as three APIs for certain key operations.
- In general, the API chosen should match the part of the kernel where it is being used, which in turn is dictated by what your code is attempting to do.

Mach:

- Mach manages processor resources such as CPU usage and memory, handles scheduling, provides memory protection, and provides a messaging-centered infrastructure to the rest of the operating-system layers.
- The Mach component provides:
  - untyped interprocess communication (IPC)
  - remote procedure calls (RPC)
  - scheduler support for symmetric multiprocessing (SMP)
  - support for real-time services
  - virtual memory support
  - support for pagers
  - modular architecture

BSD:

- Above the Mach layer, the BSD layer provides “OS personality” APIs and services.
- The BSD layer is based on the BSD kernel, primarily FreeBSD.
- The BSD component provides:
  - file systems
  - networking (except for the hardware device level)
  - UNIX security model
  - syscall support
  - the BSD process model, including process IDs and signals
  - FreeBSD kernel APIs
  - many of the POSIX APIs
  - kernel support for pthreads (POSIX threads)

Networking:

- MAC OS networking takes advantage of BSD’s advanced networking capabilities to provide support for modern features, such as Network Address Translation (NAT) and firewalls.

- The networking component provides:
    - 4.4BSD TCP/IP stack and socket APIs
    - support for both IP and DDP (AppleTalk transport)
    - multihoming
    - routing
    - multicast support
    - server tuning
    - packet filtering
    - Mac OS Classic support (through filters)

File Systems:

- MAC OS provides support for numerous types of file systems, including HFS, HFS+, UFS, NFS, ISO 9660, and others.
- The default file-system type is HFS+; MAC OS boots (and “roots”) from HFS+, UFS, ISO, NFS, and UDF. Advanced features of MAC OS file systems include an enhanced Virtual File System (VFS) design.
- VFS provides for a layered architecture (file systems are stackable).
- The file system component provides:
    - UTF-8 (Unicode) support
    - Increased performance over previous versions of Mac OS.

I/O Kit:

- The I/O Kit provides a framework for simplified driver development, supporting many categories of devices.
- The I/O Kit features an object-oriented I/O architecture implemented in a restricted subset of C++.
- The I/O Kit framework is both modular and extensible.
- The I/O Kit component provides:
    - true plug and play
    - dynamic device management
    - dynamic (“on-demand”) loading of drivers
    - power management for desktop systems as well as portables
    - multiprocessor capabilities

Q-11 Define UINavigationController. How do you navigate to another view? Explain briefly with example.

- The UINavigationController class implements a specialized view controller that manages the navigation of hierarchical content.
- This navigation interface makes it possible to present your data efficiently and makes it easier for the user to navigate that content.
- You generally use this class as-is but you may also subclass to customize the class behavior.

- The screens presented by a navigation interface typically mimic the hierarchical organization of your data.
- At each level of the hierarchy, you provide an appropriate screen (managed by a custom view controller) to display the content at that level.

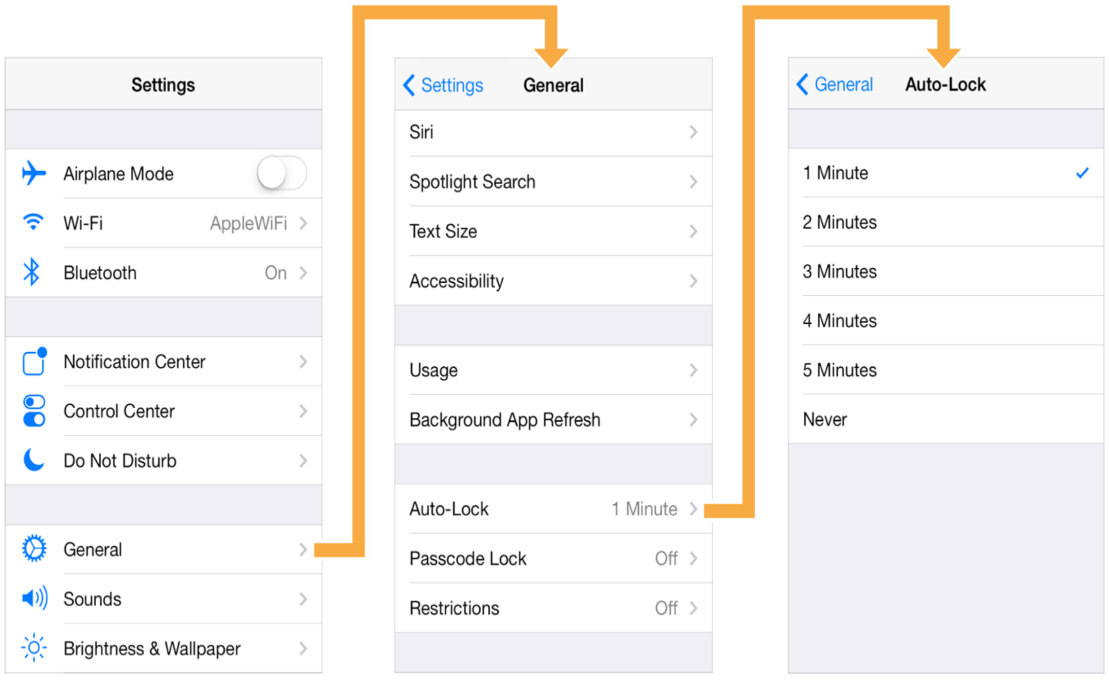

Fig. 5: Sample Navigation Interface

- Figure 5 shows an example of the navigation interface presented by the Settings application in iOS Simulator.
- The first screen presents the user with the list of applications that contain preferences.
- Selecting an application reveals individual settings and groups of settings for that application.
- Selecting a group yields more settings and so on. For all but the root view, the navigation controller provides a back button to allow the user to move back up the hierarchy.
- A navigation controller object manages the currently displayed screens using the navigation stack, which is represented by an array of view controllers.
- The first view controller in the array is the root view controller.
- The last view controller in the array is the view controller currently being displayed.
- You add and remove view controllers from the stack using segues or using the methods of this class.
- The user can also remove the topmost view controller using the back button in the navigation bar or using a left-edge swipe gesture.
- The navigation controller manages the navigation bar at the top of the interface and an optional toolbar at the bottom of the interface.
- The navigation bar is always present and is managed by the navigation controller

itself, which updates the navigation bar using the content provided by the view controllers on the navigation stack.

- When the isToolbarHidden property is false, the navigation controller similarly updates the toolbar with contents provided by the topmost view controller.
- A navigation controller coordinates its behavior with its delegate object.
- The delegate object can override the pushing or popping of a view controller, provide custom animation transitions, and specify the preferred orientation for the navigation interface.
- The delegate object you provide must conform to the UINavigationControllerDelegate protocol.

Q-12 Explain CFSocket and Berkeley Sockets in detail.

CFSocket

- A CFSocket is a communications channel implemented with a BSD socket.
- For most uses of this API, you will need to include three headers:

```
#include <CoreFoundation/CoreFoundation.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

- CFSocket can be created from scratch with CFSocketCreate(_ : _ : _ : _ : _ : _ : _ :) and CFSocketCreateWithSocketSignature(_ : _ : _ : _ : _ :) .
- You can create a CFSocket and connect simultaneously to a remote host by calling CFSocketCreateConnectedToSocketSignature(_ : _ : _ : _ : _ : _ :).
- To listen for message, you need to create a run loop source with CFSocketCreate RunLoopSource(_ : _ : _ :) and add it to a run loop with CFRunLoopAddSource(_ : _ : _ :).
- You can select the types of activities, such as connection attempts or data arrivals, that cause the source to fire and invoke your CFSocket's callback function.
- To send data, you store the data in a CFData and call CFSocketSendData(_ : _ : _ : _ :).
- When you are through with the socket, you must close it by calling CFSocketInvalidate.

Berkeley Sockets

- Berkeley sockets is also known as UNIX domain sockets or BSD sockets.
- For inter-process communication on iOS based on Berkeley sockets, following is requirement:
    - A communication channel
    - A client-server connection architecture
    - Non-blocking communication (i.e. Asynchronous I/O)
    - Message framing for data that is sent through the channel
    - Support for complex data to be sent through the channel
    - Secure encoding and decoding of data on each end of the channel
    - Error and invalidation handling
- We can create BSD socket by:

```
dispatch_fd_t fd = socket(AF_UNIX, SOCK_STREAM, 0);
```

- AF_UNIX creates a UNIX domain socket (BSD) and SOCK_STREAM makes the socket a stream socket (other options could be datagram or raw).
- For our hosts to communicate with each other we need a particular architecture where we can make sure that multiple hosts can connect to each other.
- In order to achieve this, we use a client-server architecture where one host acts as the server by creating the socket on disk and waiting for connections and the other host(s) act as clients by connecting to the server.
- The server binds to the socket and listens for connections while the client connects.
- For the client to fully connect, the server has to accept the connection.
- We can construct the socket path by appending a unique identifier to the path of the application group container directory that all applications in the group have read/write access to.
- As long as both server and clients agree on the unique identifier, we now have a channel through which they can communicate.
- First, on the server side we would start the connection by doing:

```
const char *socket_path = ...
dispatch_fd_t fd = socket(AF_UNIX, SOCK_STREAM, 0);
struct sockaddr_un addr;
memset(&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
unlink(socket_path);
strncpy(addr.sun_path, socket_path, sizeof(addr.sun_path) - 1);
bind(fd, (struct sockaddr *)&addr, sizeof(addr));
listen(fd, kLLBSDServerConnectionsBacklog);
```

- Similarly, on the client side we would connect to the server with:

```
const char *socket_path = ...
dispatch_fd_t fd = socket(AF_UNIX, SOCK_STREAM, 0);
struct sockaddr_un addr;
memset(&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
strncpy(addr.sun_path, socket_path, sizeof(addr.sun_path) - 1);
connect(fd, (struct sockaddr *)&addr, sizeof(addr));
```

Q-13 How Cocoa Objects Manage Memory. OR How to achieve memory management in iOS? Explain with example.

- Memory management is the programming discipline of managing the life cycles of objects and freeing them when they are no longer needed.
- Managing object memory is a matter of performance; if an application doesn’t free unneeded objects, its memory footprint grows and performance suffers.
- Memory management in a Cocoa application that doesn’t use garbage collection is based on a reference counting model.

- When you create or copy an object, its retain count is 1.
- Thereafter other objects may express an ownership interest in your object, which increments it's retain count.
- The owners of an object may also relinquish their ownership interest in it, which decrements the retain count.
- When the retain count becomes zero, the object is DE allocated (destroyed).
- An object can hardly be an element of a collection if that object can go out of existence at any time; so when you add an element to a collection, the collection asserts ownership of the object by retaining it.
- Thereafter, the collection acts as a well-behaved owner.
- If this is a mutable collection, then if an element is removed from it, the collection releases that element.
- If the collection object goes out of existence, it releases all its elements.
- Removing an object from a mutable collection constituted a potential trap.

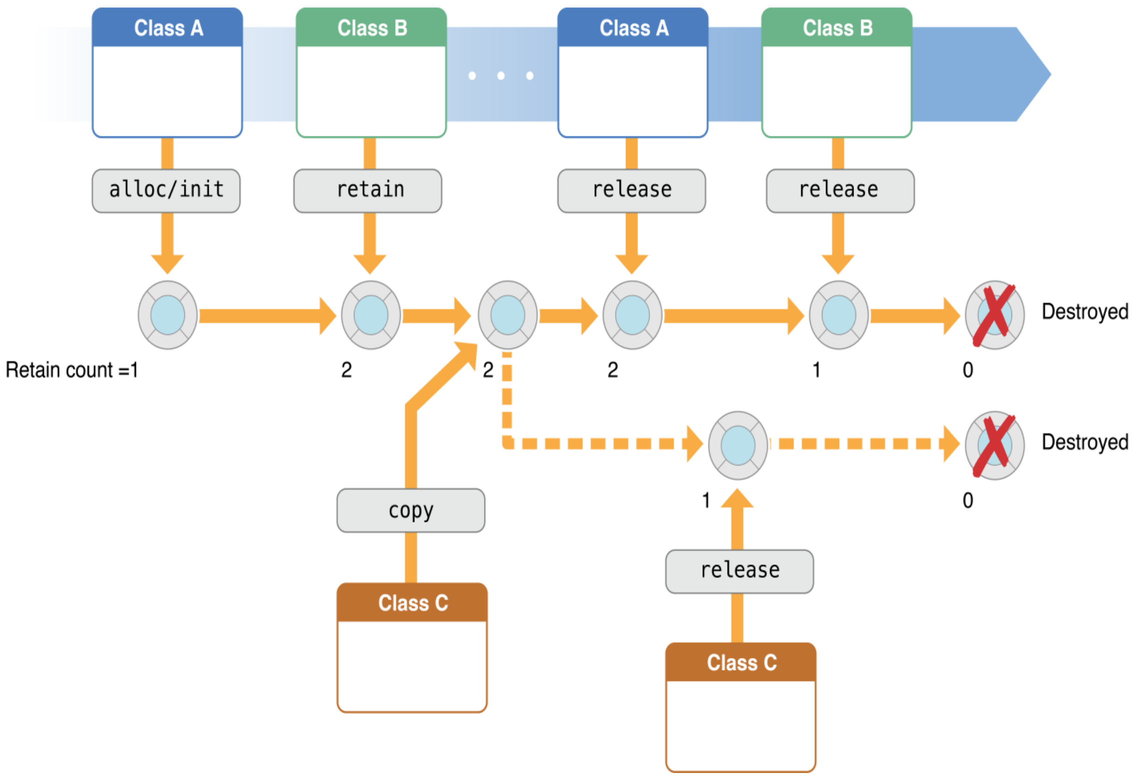

Fig. 6: Memory management

- Memory management can be achieved following three concepts:

Autorelease pools:

- Sending autorelease to an object marks the object for later release, which is useful when you want the released object to persist beyond the current scope.
- Autoreleasing an object puts it in an autorelease pool (an instance of NSAutoreleasePool), which is created for an arbitrary program scope.
- When program execution exits that scope, the objects in the pool are released.

Deallocation:

- When an object’s retain count drops to zero, the runtime calls the dealloc method of the object’s class just before it destroys the object.
- A class implements this method to free any resources the object holds, including objects pointed to by its instance variables.

Factory methods:

- Many framework classes define class methods that, as a convenience, create objects of the class for you.
- These returned objects are not guaranteed to be valid beyond the receiving method’s scope.

Example:

- Consider the following Objective-C code:

```
id obj = myMutableArray[0]; // an NSMutableArray
[myMutableArray removeObjectAtIndex: 0]; // bad idea in non-ARC code!
// ... could crash here by referring to obj ...
```

- When you remove an object from a mutable collection, the collection releases it.
- So, the second line of that code involves an implicit release of the object that used to be element 0 of myMutableArray.
- If this reduces the object’s retain count to zero, it will be destroyed.
- The pointer obj will then be a dangling pointer, and a crash may be in our future when we try to use it as if it were a real object.
- With ARC, however, that sort of danger doesn’t exist. Assigning a reference type object to a variable retains it!
- But we did assign this object to a variable, obj, before we removed it from the collection. Thus that code is perfectly safe, and so is its Swift equivalent:

```
let obj = myMutableArray[0]
myMutableArray.removeObject(at:0)
// ... safe to refer to obj ...
```

- The first line retains the object. The second line releases the object, but that release balances retain that was placed on the object when the object was placed in the collection originally.
- Thus the object’s retain count is still more than zero, and it continues to exist for the duration of this code.

Q-14 Explain iPhone Architecture in detail.

- The iPhone architecture is shown in figure 7.

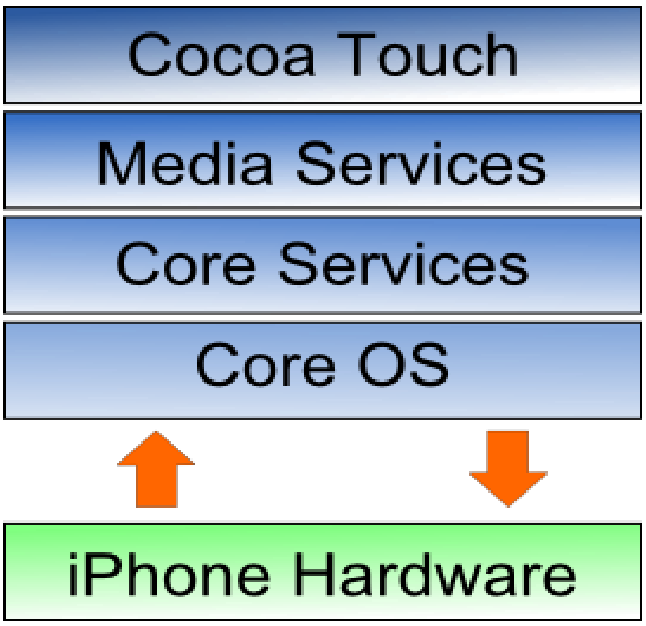

Fig. 7: IPhone Architecture

Cocoa Touch Layer

- It is a top layer of the iPhone OS stack and it contains the frameworks that are most commonly used by iPhone application developers.
- Cocoa Touch is primarily written in Objective-C, and it is based on the standard Mac OS X Cocoa API.
- Some of the main features and technologies of Cocoa Touch are:
    - App Extension
    - Handoff
    - Document Picker
    - AirDrop
    - TextKit
    - UIKit Dynamics
    - Multitasking
    - Auto Layout
    - Storyboards
    - UI State Preservation, etc.
- Cocoa Touch provides the key frameworks for developing applications on devices running iOS. Some of these key frameworks are:
    - Foundation Kit Framework
    - UIKit Framework (based on Application Kit)
    - GameKit Framework
    - iAd Framework
    - MapKit Framework
    - Address Book UI Framework
    - Twitter Framework, etc.

Media Layer

- It is the second layer from the top of the stack. It provides the iPhone OS with audio, video, animation and graphics capabilities.
- As with the other layers of the iPhone OS stack, the Media layer comprises a number of

frameworks that can be utilized when developing iPhone apps.

- The media layer comprises of following frameworks:
    - Core Graphics Framework
    - Quartz Core Framework
    - OpenGL ES framework
    - AV Foundation framework
    - Core Audio Frameworks
    - Media Player framework

Core Services Layer

- It is the third layer from the top of the stack.
- The iPhone Core Services layer provides much of the foundation on which the above layers are built.
- The core services layer comprises of following frameworks:
    - Address Book framework
    - Core Data Framework
    - Core Foundation Framework
    - Foundation Framework
    - Core Location Framework
    - Store Kit Framework

Core OS Layer

- The Core OS Layer is the bottom layer of the iPhone OS stack and sits directly on top of the device hardware.
- This layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.
- The core OS layer comprises of following frameworks:
    - CFNetwork framework
    - External Accessory framework
    - Security Framework

iPhone Hardware

- Hardware devices are managed by iPhone OS and provides the technologies needed for implementing native applications on the phone.
- The OS ships with several system applications such as Mail, Safari, and Phone that provide standard services to the user.

Q-15 Explain how to create outlets in interface builder with example.

- You can use Interface Builder to create the outlets for the UI components that the app interacts with programmatically.
- A common naming convention is to use the UI component's class name without the UI class prefix at the end of an outlet property's name—for example, billAmountLabel rather than billAmountUILabel.
- Interface Builder makes it easy for you to create outlets for UI components by control dragging from the component into your source code.
- To do this, you'll take advantage of the Xcode Assistant editor.

Example:

- Consider that you have viewcontroller with one label and You have to create outlet for this label.
- To create outlets, ensure that your scene’s storyboard is displayed by selecting it in the Project navigator.
- Next, select the Assistant editor button on the Xcode toolbar (or select View > Assistant Editor > Show Assistant Editor).
- Xcode’s Editor area splits and the file ViewController.swift is displayed to the right of the storyboard.
- By default, when viewing a storyboard, the Assistant editor shows the corresponding view controller’s source code.
- However, by clicking Automatic in the jump bar at the top of the Assistant editor, you can select from options for previewing the UI for different device sizes and orientations, previewing localized versions of the UI or viewing other files that you’d like to view side-by-side with the content currently displayed in the editor.
- You’ll now create an outlet for the Label that displays the user’s input.
- You need this outlet to programmatically change the Label’s text to display the input in currency format.
- Outlets are declared as properties of a view controller class. To create the outlet:
    - Control drag from the Label to below the Class of viewcontroller in ViewController.swift and release. This displays a popover for configuring the outlet.
    - In the popover, ensure that Outlet is selected for the Connection type, specify the name billAmountLabel for the outlet’s Name and click Connect.
- Xcode inserts the following property declaration in class ViewController:

  ```
  @IBOutlet weak var billAmountLabel: UILabel!
  ```
- You can now use this property to programmatically modify the Label’s text.
- Repeat the above steps to create outlets for the other UI components if needed during programming.