**Subjective Que/Ans**

**1: How to save model class in NSUserDefault?**

Ans:

**This is the model class** Users

@interface Users : NSObject

@property (nonatomic, strong) NSString \*email;

@property (nonatomic, strong) NSString \*name;

@property (nonatomic, strong) NSString \*mobileNo;

**This is the code for save it in userdefault:-**

Users \*userData = [Users new];

userData.email = @”[jschoudahry01@gmail.com](mailto:jschoudahry01@gmail.com)”;

userData.name = @”Jogendar”;

userData.mobileNo = @”9772094100”;

NSData \*encodedObject = [NSKeyedArchiver archivedDataWithRootObject:userData];

[[NSUserDefaults standardUserDefaults] setObject:encodedObject forKey:kProfileData];

[[NSUserDefaults standardUserDefaults]synchronize];

**2: Describe how view controller lifecycle order when another view controller start over current view controller?**

**Ans:-**

* loadView
* viewDidLoad
* viewWillAppear
* viewWillLayoutSubviews
* viewDidLayoutSubviews
* viewDidAppear

Let me explain all those stages.

**loadView**

This event creates the view that the controller manages. It is only called when the view controller is created programmatically. This makes it a good place to create your views in code.

**viewDidLoad**

The viewDidLoad event is only called when the view is created and loaded into memory. But the bounds for the view are not defined yet. This is a good place to initialize the objects that the view controller is going to use.

**viewWillAppear**

This event notifies the view controller whenever the view appears on the screen. In this step the view has bounds that are defined but the orientation is not set.

**viewWillLayoutSubviews**

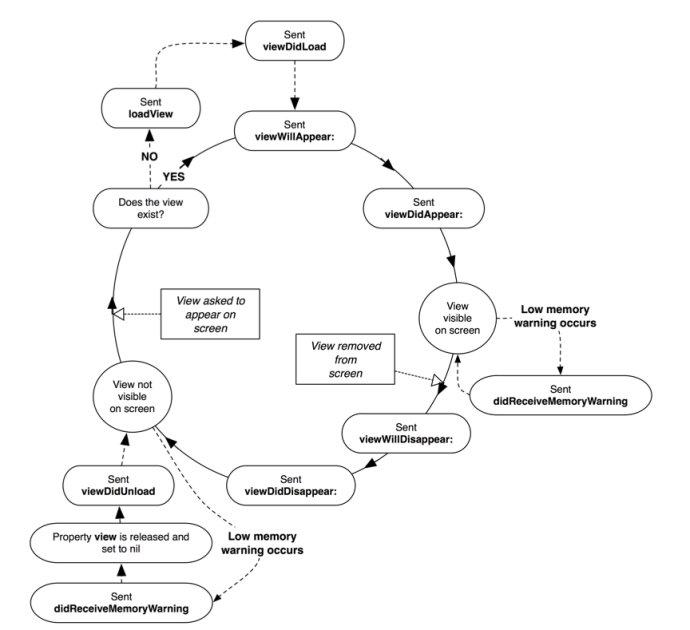
This is the first step in the lifecycle where the bounds are finalized. If you are not using constraints or Auto Layout you probably want to update the subviews here.

**viewDidLayoutSubviews**

This event notifies the view controller that the subviews have been setup. It is a good place to make any changes to the subviews after they have been set.

**viewDidAppear**

The viewDidAppear event fires after the view is presented on the screen. Which makes it a good place to get data from a backend service or database.



**3: How to support multi-language and multi resolution ?**

**Ans:-**

Step 1 :

click on project -> info -> scroll down -> localizations -> click on '+' and add your required language -> check as required -> Finish.

Step 2 :

click on 'Supporting Files' -> right click -> New File -> select Resource on left-side list -> select 'Strings Files' -> next -> name it 'Localizable' -> create

Step 3 :

click on newly created file -> go to Utilities -> file inspector -> click on 'Localize..' -> check all language

Step 4 :

open Localizable.strings(English) write "help" = "I can't help you";

here "help" is a key "I can't help you" is a value. You may change them as your wish.

open Localizable.strings(French) or the language you have set. write accordingly and you can translate "I can't help you" to any said language.

Step 5 :

// for test NSString \*str = NSLocalizedString(@"help", Nil);   
NSLog(@"%@",str);

**For Multi resolution: We can use auto-layout and stack view for same.**

**4:Explain error handling and exception management ?**

**Ans:**

**Error Handling:**

When an error is thrown, some surrounding piece of code must be responsible for handling the error—for example, by correcting the problem, trying an alternative approach, or informing the user of the failure.

There are four ways to handle errors in Swift. You can propagate the error from a function to the code that calls that function, handle the error using a do-catch statement, handle the error as an optional value, or assert that the error will not occur.

**Exception:**

Exceptions cause applications to crash if left unhandled. They generally occur when trying to perform an operation on an object incorrectly, such as using an out-of-bounds index to access an array item, or passing nil to a method that doesn’t accept it. In other words, they are caused by developer mistakes.

You shouldn’t ever need to throw exceptions in you application, however they can be useful if you’re writing a library – be it for your app or for other developers. If someone calls a method in your library with invalid arguments, throwing an exception is a way of telling them they’re doing it wrong. In case you ever need to do this, exceptions can be created with the NSException class factory method shown below which takes the 3 key bits of information to help identify and debug the problem.