# Coding Guidelines | iOS

https://swift.org/documentation/api-design-guidelines/

***General:***

Use Core data for supporting offline data. If all the data need not to be supported offline make the core data part of model layer not a separate layer.

Don't make unnecessary API/database calls. Use caching for less frequent changing data.

Use meaning full inheritance. Make a base class for something which is reusable across many subclasses.

Use singleton when its necessary to bond number of that object of that class to one not because you want to share something across application.

Method which modify a class object preferred to be placed in a category not in utility. e.g. string truncation, device info, etc... should be in string category not in utility.

Avoid making large static class (classes with no instance variable and only class method) as they are just C method in disguise of obj

C. Don't hard code any values inside the code. Move them to properties / constants files or strings file. This makes it easy to manage and change.

Don't write large methods. Break it logically. Write utility methods if you think something can be reused inside application or across. Make sure the names of classes/variables/methods are self explaining (for ex: isActive, lastName, userByID() etc). not like MyAppSingleton, as app may have many singleton and nobody can guess what this particular class suppose to do. Follow proper naming convention.

rol.html)





https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingMethods.html

Access control (https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift\_Programming\_Language/AccessCont

Enable Bitcode into the application.

Delegate protocols should be class-only so they can be weakly referenced. Put all images into “Assets.xcassets” folder.

Don’t make view/viewcontroller class as singleton class. Update UI only in main thread.

Logical and physical folder structure must be same. Use proper spacing and formatting.

Write comments in case you write a complex business logic. Use Crashlytics into application.

Use “NSLocalizedString” method when we use static strings even for single language. Don't add unnecessary variables / objects. Initialize only when necessary.

Write //TODO: or //FIXME: in code wherever you see something can be added to improve or for pending tasks. This will help in finding the pending items faster and in code review.

Indent one more incase you are splitting a line. You should not have too long lines. Lets follow a limit of 100 characters per line. Follow the KISS principle. It means Keep It Simple Stupid. (<http://en.wikipedia.org/wiki/KISS_principle)>Don't write overly complicated code.

It’s best to use an Xcode storyboard for your launch screen.

***Logging:***

Write proper logs (with Logging Levels: DEBUG, ERROR, INFO, WARNING etc...) in code at different needed places.

***Error Handling:***

Do proper error handling. Use exception handling whenever you are making a network call or XML/JSON parsing or file operation where the result could be different from expected.

Handle general error in base-class and classes specific to a particular call should be forwarded to the class.

***Unit Testing:***

Use XCTest framework for unit and UI testing.

Make sure that you cover 65-70% of code using test cases. It should be incremental i.e. with every new build have equal or more code coverage.

This will reduce bugs in QA and reduce the over all effort in project development.

***Guidelines wrt Swift:***

https://swift.org/documentation/api-design-guidelines/ https://github.com/raywenderlich/swift-style-guide

**SwiftFormat - Swift Code Format Automation Script**

SwiftFormat is a open source code library and command-line tool for reformatting swift code.

It applies a set of rules to the formatting and space around the code, leaving the meaning intact.

Having a tool to automatically enforce a common style eliminates those issues, and lets you focus on the *operation* of the code, not its presentation.

Github: https://github.com/nicklockwood/SwiftFormat

#### How To Install :-

**Step 1 :** Install Homebrew in your home directory

for more information on this step please visit - https://brew.sh

**Step 2 :** Install swiftformat via home-brew run the following in your terminal

brew install swiftformat

Step 3 : Navigate to git root directory of your project in terminal run the following in your terminal

swiftformat .

For more information please visit https://github.com/nicklockwood/SwiftFormat