**Swift Language Interview Basics**

**-: KeyWords :-**

* Keywords used in declarations: associatedtype, class, deinit, enum, extension, fileprivate, func, import, init, inout, internal, let, open, operator, private, precedencegroup, protocol, public, rethrows, static, struct, subscript, typealias, and var.
* Keywords used in statements: break, case, catch, continue, default, defer, do, else, fallthrough, for, guard, if, in, repeat, return, throw, switch, where, and while.
* Keywords used in expressions and types: Any, as, catch, false, is, nil, rethrows, self, Self, super, throw, throws, true, and try.
* Keywords used in patterns: \_.
* Keywords that begin with a number sign (#): #available, #colorLiteral, #column, #dsohandle, #elseif, #else, #endif, #error, #fileID, #fileLiteral, #filePath, #file, #function, #if, #imageLiteral, #keyPath, #line, #selector, #sourceLocation, and #warning.
* Keywords reserved in particular contexts: associativity, convenience, didSet, dynamic, final, get, indirect, infix, lazy, left, mutating, none, nonmutating, optional, override, postfix, precedence, prefix, Protocol, required, right, set, some, Type, unowned, weak, and willSet. Outside the context in which they appear in the grammar, they can be used as identifiers.

**Literals : -**

A literal doesn’t have a type on its own. Instead, a literal is parsed as having infinite precision and Swift’s type inference attempts to infer a type for the literal. For example, in the declaration let x: Int8 = 42, Swift uses the explicit type annotation (: Int8) to infer that the type of the integer literal 42 is Int8. If there isn’t suitable type information available

*literal* → [numeric-literal](https://docs.swift.org/swift-book/ReferenceManual/LexicalStructure.html#grammar_numeric-literal) | [string-literal](https://docs.swift.org/swift-book/ReferenceManual/LexicalStructure.html#grammar_string-literal) | [boolean-literal](https://docs.swift.org/swift-book/ReferenceManual/LexicalStructure.html#grammar_boolean-literal) | [nil-literal](https://docs.swift.org/swift-book/ReferenceManual/LexicalStructure.html#grammar_nil-literal)

*numeric-literal* → **-***opt* [integer-literal](https://docs.swift.org/swift-book/ReferenceManual/LexicalStructure.html#grammar_integer-literal) | **-***opt* [floating-point-literal](https://docs.swift.org/swift-book/ReferenceManual/LexicalStructure.html#grammar_floating-point-literal)

*boolean-literal* → **true** | **false**

*nil-literal* → **nil**

Special characters can be included in string literals of both the single-line and multiline forms using the following escape sequences:

* Null character (\0)
* Backslash (\\)
* Horizontal tab (\t)
* Line feed (\n)
* Carriage return (\r)
* Double quotation mark (\")
* Single quotation mark (\')
* Unicode scalar (\u{*n*}), where *n* is a hexadecimal number that has one to eight digits

**TYPES : -**

GRAMMAR OF A TYPE

*type* → [function-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_function-type)

*type* → [array-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_array-type)

*type* → [dictionary-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_dictionary-type)

*type* → [type-identifier](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_type-identifier)

*type* → [tuple-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_tuple-type)

*type* → [optional-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_optional-type)

*type* → [implicitly-unwrapped-optional-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_implicitly-unwrapped-optional-type)

*type* → [protocol-composition-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_protocol-composition-type)

*type* → [opaque-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_opaque-type)

*type* → [metatype-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html" \l "grammar_metatype-type)

*type* → [any-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_any-type)

*type* → [self-type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_self-type)

*type* → **(** [type](https://docs.swift.org/swift-book/ReferenceManual/Types.html#grammar_type) **)**

**Statements**

GRAMMAR OF A STATEMENT

*statement* → [expression](https://docs.swift.org/swift-book/ReferenceManual/Expressions.html#grammar_expression) **;***opt*

*statement* → [declaration](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_declaration) **;***opt*

*statement* → [loop-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_loop-statement) **;***opt*

*statement* → [branch-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_branch-statement) **;***opt*

*statement* → [labeled-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html" \l "grammar_labeled-statement) **;***opt*

*statement* → [control-transfer-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_control-transfer-statement) **;***opt*

*statement* → [defer-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_defer-statement) **;***opt*

*statement* → [do-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_do-statement) **;***opt*

*statement* → [compiler-control-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_compiler-control-statement)

*statements* → [statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_statement) [statements](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_statements) *opt*

Loop statements allow a block of code to be executed repeatedly, depending on the conditions specified in the loop. Swift has three loop statements: a for-in statement, a while statement, and a repeat-while statement.

Control flow in a loop statement can be changed by a break statement and a continue statement and is discussed in [Break Statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#ID441) and [Continue Statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#ID442) below.

GRAMMAR OF A LOOP STATEMENT

*loop-statement* → [for-in-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_for-in-statement)

*loop-statement* → [while-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_while-statement)

*loop-statement* → [repeat-while-statement](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_repeat-while-statement)

A for-in statement has the following form:

1. for item in collection {
2. statements
3. }

A for-in statement has the following form:

1. for item in collection {
2. statements
3. }

GRAMMAR OF A WHILE STATEMENT

*while-statement* → **while** [condition-list](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_condition-list) [code-block](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_code-block)

*condition-list* → [condition](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_condition) | [condition](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_condition) **,** [condition-list](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_condition-list)

*condition* → [expression](https://docs.swift.org/swift-book/ReferenceManual/Expressions.html#grammar_expression) | [availability-condition](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_availability-condition) | [case-condition](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_case-condition) | [optional-binding-condition](https://docs.swift.org/swift-book/ReferenceManual/Statements.html#grammar_optional-binding-condition)

*case-condition* → **case** [pattern](https://docs.swift.org/swift-book/ReferenceManual/Patterns.html#grammar_pattern) [initializer](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_initializer)

*optional-binding-condition* → **let** [pattern](https://docs.swift.org/swift-book/ReferenceManual/Patterns.html#grammar_pattern) [initializer](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_initializer) | **var** [pattern](https://docs.swift.org/swift-book/ReferenceManual/Patterns.html#grammar_pattern) [initializer](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_initializer)

A repeat-while statement allows a block of code to be executed one or more times, as long as a condition remains true.

A repeat-while statement has the following form:

1. repeat {
2. statements
3. } while condition

**------------------------------------------------------------------**

GRAMMAR OF A REPEAT-WHILE STATEMENT

*repeat-while-statement* → **repeat** [code-block](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_code-block) **while** [expression](https://docs.swift.org/swift-book/ReferenceManual/Expressions.html#grammar_expression)

---------------------------------------------------------------------------------------------------------

GRAMMAR OF A REPEAT-WHILE STATEMENT

*repeat-while-statement* → **repeat** [code-block](https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#grammar_code-block) **while** [expression](https://docs.swift.org/swift-book/ReferenceManual/Expressions.html#grammar_expression)

**------------------------------------------------------------------**

1. if condition {
2. statements
3. }

------------------------------------------------------------------

1. if condition {
2. statements to execute if condition is true
3. } else {
4. statements to execute if condition is false
5. }
6. if condition {
7. statements to execute if condition is true
8. } else {
9. statements to execute if condition is false
10. }
11. if condition {
12. statements to execute if condition is true
13. } else {
14. statements to execute if condition is false
15. }

Guard Statement

A guard statement is used to transfer program control out of a scope if one or more conditions aren’t met.

A guard statement has the following form:

1. guard condition else {
2. statements
3. }

The else clause of a guard statement is required, and must either call a function with the Never return type or transfer program control outside the guard statement’s enclosing scope using one of the following statements:

* return
* break
* continue
* throw

Switch Statement

A switch statement allows certain blocks of code to be executed depending on the value of a control expression.

A switch statement has the following form:

1. switch control expression {
2. case pattern 1:
3. statements
4. case pattern 2 where condition:
5. statements
6. case pattern 3 where condition,
7. pattern 4 where condition:
8. statements
9. default:
10. statements
11. }

# Application life cycle in iOS

* The application life cycle constitutes the sequence of events that occurs between the launch and termination of application.
* It is very important to understand for all the iOS Developers, who wants smooth user experience.

## Steps involved from device reboot to app launch:-

* When user turn on the phone, no applications are running except app which belong to OS.When user tap on your app icon, SpringBoard launches your app.

*AppDelegate is the application delegate object. It inherits the UIResponder class and implements the UIApplicationDelegate delegate protocol*

* Main entry into iOS apps is UIApplicationDelegate**.**Itis a protocol and you need to implement that into your app to get notified about user events such as app launch, app goes into background or foreground, app is terminated, a push notification was opened etc.
* UIResponder class make AppDelegate have the ability to respond to user events and UIApplicationDelegate enables the AppDelegate to be an application delegate object to manage and respond to the life cycle of the application.

## Execution States for Apps:-

1. **Not Running** **state** : The app has not been launched or terminated by the system.
2. **Inactive** **state** : The app is entering the foreground state but not receiving events.
3. **Active** **state**: The app enters the foreground state and can process event.
4. **Background** **state** : In this state, if there is executable code, it will execute and if there is no executable code or the execution is complete, the application will be suspended immediately.
5. **Suspended** **state** : The app is in the background(in memory) but is not executing code and if system does not have enough memory, it will terminate the app.
6. 



