Question and Answer

1. What is the difference between var and let?

Answer: Both let and var are used to create variable in swift. The key work let used to create immutable variable, Once the variable is created with let keyword, the variable cannot reassign it. With the var keywork, we can create the variable and reassign it.

2. What is an optional?

Answer: An optional is a type that represent either a wrapped value or nil, the absence of a value. To make a parameter optional, the question sign (?) or key word Optional are used. Here is the example.

```
let shortForm: Int? = Int("42")
```

Here, the variable shortForm is declared with let keywork and type is integer and make it optional with question sign (?). The alternative way is to make the variable optional by using Optional keywork. Here is the example,

```
let longForm: Optional<Int> = Int("42")
```

3. What is optional chaining vs optional binding?

Answer: Optional chaining is a process for querying and calling properties, methods, and subscripts on an optional that might currently be nil. If the optional contains a value, the property, method, or subscript call succeeds; if the optional is nil, the property, method, or subscript call returns nil. Multiple queries can be chained together, and the entire chain fails gracefully if any link in the chain is nil.

For example, let declare two class named Person and Residence.

```
class Person {
      var residence: Residence?
      }

  class Residence {
       var numberOfRooms = 1 }
```

The class Person has variable named residence and it is referring to next class Residence and the residence variable is optional. The Residence class has an integer property numberOfRooms. According to the definition of optional chaining to call class by using option chaining, the instance of the class needs to be created then use if and else statement.

```
let john = Person()
if let roomCount = john.residence?.numberOfRooms {
print("John's residence has \(roomCount) room(s).")
} else {
print("Unable to retrieve the number of rooms.")
}
```

The print statement will print total room for Jon. This is example of using option chaining by using class.

optional binding is a simpler and recommended way to unwrap an optional. You use optional binding to check if the optional contains a value or not. If it does contain a value, unwrap it and put it into a temporary constant or variable. For example …

```
var stockCode:String? = findStockCode("Facebook")
let text = "Stock Code - "
if let tempStockCode = stockCode {
  let message = text + tempStockCode
  println(message)
}
```

If stockCode contains a value, unwrap it, set its value to tempStockCode and execute the conditional block.

4. What are the different ways to unwrap an optional? How do they work? Are they safe?

Answer: The are seven ways to unwrap an optional in swift.

- Forced unwrapping: To make a parameter optional, the exclamatory sign is used.

  var x : String? = “Test”

```
let  a:String  =  x!
```

Here the unwrapping x could crush if x is nil. There for it not safe.

- Implicitly unwrapped: The implicitly unwrapped variable declaration is unsafe in many cases. Here is the syntax to declare a variable as implicitly.

  ```
  var  a  =  x!
  ```

- Optional Binding: It is safe approach compare forced and implicitly unwrapping because with the optional binding start with if statement which is Boolean. if the statement is true then will execute the next line of code otherwise it will return false. Here is the example.

  ```
  if let a = x {
  print ("x was successfully unwrapped and is = \(a)")
  }
  ```

- Optional Chaining: It is also safe approach. Here is the example.

  ```
  let a = x?.count
  ```

- Nil coalescing operator: It is also safe approach. Here is the example.

  ```
  let  A =  x ?? ""
  ```

- Guard statement: It is similar with if statement and it also safe. Here is the example.

  ```
  guard let a = x  else {
  return
  }
  ```

- Optional pattern: Safe. Here is the example.

  ```
  if  case let a?  = x {

  print (a)
  }
  ```

5. What is a closure?

Answer: Closures are self-contained block of code or functionality that can be passed inside the method or outside of the method. It also known as call-back function. The closure consists with parameters, return type and the statement. Here is the syntax for declaring closures.

```
{ ( parameters ) -> return type in
statements
}
```

Here is the example of the closure.

```
reversedNames = names.sorted(by: { (s1: String, s2: String) -> Bool
in
return s1 > s2
})
```

6. What is the difference between a class and a struct?

Answer: In swift structs are value types and classes are reference types. A copy of the struct contains two unique copies of the data. The copy of the class contains two reference of the one instance data. The structs are declared with key word struct and classes are declared with the key word class followed by the name of the class.

7. What is the syntax '??' do?

Answer: This is a swift option ternary operator and also forced unwraps an optional to access the value wrapped inside a when a is not nil, and to return b otherwise. Here is the example.

```
// program to check pass or fail
let marks = 60

// use of ternary operator
```

let result = (marks >= 40) ? "pass" : "fail"

print("You " + result + " the exam")

8. What is a tuple?

Answer: Tuple is a group of different values represented as one. According to apple, **a tuple type is a comma-separated list of zero or more types, enclosed in parentheses**. It's a miniature version of a struct. Here is example of creating tuple.

```
var point = (0, 0)

point.0 = 10
point.1 = 15

point // (10, 15)
```

9. What is Any vs AnyObject?

Answer: In swift Any can represent an instance of any type at all, including function types and optional types and other hand AnyObject can represent an instance of any class type.

10. What is a protocol?

Answer: The protocol defines a blueprint of methods or properties that can be used or called by the classes or other type. The key word protocol is used to define a protocol. Here is the example.

protocol Greet {

var name : String { get }

func message ( )

}

Here is the implementation of the class with protocol.

```
class Employee : Greet  {

var  name = “Mohammad”

func message  ( ) {
print (“Good Morning “ , name)

}
```

11.What is Delegation?

Answer: Delegation is a simple and powerful pattern in which one object in a program act on behalf of, or in coordination with, another object. The delegating object keeps a reference to the other object–the delegate–and at the appropriate time sends a message to it. The message informs the delegate of an event that the delegating object is about to handle or has just handled. The delegate may respond to the message by updating the appearance or state of itself or other objects in the application, and in some cases, it can return a value that affects how an impending event is handled. The main value of delegation is that it allows you to easily customize the behaviour of several objects in one central object