**Question 1:**

**import** UIKit

**func** sum(n1: Int, n2: Int) ->(Int){

**return** n1 + n2

}

print(sum(n1: 5,n2: 10))

**Question 2:**

//let n1: Int = 1

//let n2: Float = 2.0

//let n3: Double = 3.34

//Error message: Type of expression is ambiguous => n1,n2,n3 have de different type

//var result = n1 + n2 + n3

// Correction

**let** n1: Double = 1

**let** n2: Double = 2.0

**let** n3: Double = 3.34

**var** result = n1 + n2 + n3

print(result)

**Question 3 :**

init() function allows to initialize the properties of the class, it can be overload

Question 4 :

Protocol is a class that contains only the methods without body. And when a class inherit a protocol, we need to define those methods in the protocol

**Question 5:**

The double question mark operator is also called an nil-coalescing operator, if the left value is nil then use the right value, if this left value is not nil then use the left value

**var** anOptional: Int? = 17

**let** aValue = 5

**var** theResult: Int

theResult = anOptional ?? aValue

print(theResult). // theResult = 17 and if anOptional: Int? Then theResult = 5

**Question 6 :**

Func testGuard(){

Guard expression else{

//code

return

}

}

If expression = true then do not execute the statement inside the guard

If expression = false then execute the statement inside the guard

Guard should be inside of the loop or of the func, and should have the return value or throw error

We can have the multiple condition in guard

Guard condition1, condition2, condition3, else{

// body

}

If one of condition is false then the body is executed

**Question 7 :**

There are 3 primary collections in swift:

- Array :

var Arr: [Int] = []

- Set :

var letters = Set<Character>()

- Dictionary

var names: [Int: String] = [:]

**Question 8:**

Structures are value types and classes are reference type, then Structures are in Stack memory and Classes are in heap memory (RAM), and then Structure is faster than Classes.

**Question 9 :**

Optional chaining is a process for calling a methods, properties that could be nil. The properties or methods return nil or the value not nil

**class** Person {

**var** residence: Residence?

**var** age: Int?

}

**class** Residence {

**var** numberOfRooms = 1

}

**let** john = Person()

john.residence = Residence()

**if** **let** roomCount = john.residence?.numberOfRooms {

print("John's residence has \(roomCount) room(s).")

} **else** {

print("Unable to retrieve the number of rooms.")

}

**Question 10 :**

when we use an optional that has a nil, it could make an error

Then avoiding this error, we use Optional binding that consist to use if -let

Example :

Var str: String? = nil

If let name = str {

print(string)

}

**Question 11:**

In-Out parameter is a parameter that has the same name with 2 different values inside and outside function. But we have to use keyword inout, and & for using this function

Func doubleInPlace(number: inout int){

number \*= 2

}

Var myNum = 10

doubleInPlace(number: &myNum)

**Question 12:**

The default parameter can be set as below :

Func functionName(parameterName: Type = defaultValue){

//body

}

func add(a: Int = 0, b: Int = 0) {

    print(a+b)

}

add(a: 10, b: 20)

add(a: 5)

add(b: 4)

**Question 13:**

Force Unwrapping consists to convert the type Optional to another type

Example :

Let x: Int?

Let y = x! // convert Optional x to Int in the variable Y

While :

Let x: Int! = nil // is a implicit unwrapping

**Question 14:**

Variables in the Structure are not mutable( reassign a new value), then if we want to make it mutable, We need to use the keyword mutating

**struct** Student {

**var** marks = 0

//mutating method

**mutating** **func** getGrade() {

marks = marks + 300

print("New Marks :", marks)

}

}

**var** std1 = Student()

std1.marks = 600

std1.getGrade()

**Question 15:**

Before to deallocate the memory for the object(delete object), the function deinit() is execute for doing certain tasks.

// ======== deinit function =============

**class** Garage {

**var** numberOfVehicles : Int

**init**(){

numberOfVehicles = 90

print("Total Vehicles in Garage :", numberOfVehicles)

}

//deinitializer - ARC : Automatic Reference Counting

**deinit** {

print("Object is deallocated from Memory")

}

}

**var** result: Garage? = Garage() // Garage? allows to delete object with nil otherwise we can not do result = nil

result?.numberOfVehicles = 100

print(result?.numberOfVehicles)

result = **nil** // here it delete the object and deinit() is invoked before

result?.numberOfVehicles = 99

print(result?.numberOfVehicles)

**Question 16:**

Protocols is a class which have the function without the body.

Protocol can inherit multiple protocol

A class can inherit a protocol and need to override the function in protocol

Protocol can have extension

**protocol** Wish {

**var** name : String { **get** }

**func** displayMessage()

}

**Question 17 :**

Class can have the complete function with the body

Class can inherit only one class or protocol

Class doesn’t have the extension

Protocols is a class which have the function without the body.

Protocol can inherit multiple protocol

Protocol can have extension

**Question 18:**

// add return

**struct** Apple{

}

**func** pick(apple : Apple?){

**guard** **let** apple = apple **else** {

print("No apple found")

**return**

print("Apple not nil",apple)

}

**var** st: Apple? = Apple()

pick(apple: st!)

st = **nil**

pick(apple: st)

**Question 19:**

Protocol A{

}

Protocol B{

}

Protocol C{

}

class D:A,B,C{

}

**Question 20:**

**var** first = ["John", "Paul"]

**let** second = ["George","Ringo"]

first.append(contentsOf: second)

print(first)