Question #1

**func** addtwonumber(num1: Int, num2: Int)->Int

{

**let** result = num1 + num2

**return** result

}

addtwonumber(num1: 23, num2: 23)

Question #2

We have to convert all values to same data type.

Question#3

Init()  
It is used to initialize the variable and constant for the instant of class, structure or for enumeration.

Question#4

Protocols

A protocol is define as a blueprint for all derived classes. We declare variables and functions in protocol without their definition.

e.g

**protocol** MathsFu{

**var** num1: Int { **get** }

**var** num2: Int { **get** }

**func** AddNum()

}

**class** AddNumber1: MathsFu{

**var** num1: Int = 98

**var** num2: Int = 8

**func** AddNum() {

**let** result = num1 + num2

print(result)

}

}

**var** addnumber1 = AddNumber1()

addnumber1.AddNum()

in the above example I declared two variables and one function in Protocol and implement their definition in class “AddNumber1”.

Question#5

Double question marks also know as “nil Coalesing” operator. It will provide a default value in case of nil value. e.g

Var speedlimit: Int?

Var maxspeed = speedlimit ?? 80

Print(maxspeed)

Question#6

Guard Statement

Guard instruction works along with else statement & is similar to if else but only difference is code only executed when condition is false e.g.

**for** ii **in** 1...10

{

**guard** ii % 2 != 0 **else**{

print(ii)

**break**

}

}

Question #7

There are three primary collection types.

1. Array
2. Set
3. Dictionary

Question # 8

Difference between structure and classes is structure work as value type and classes are work as reference type.

Question # 9

Optional chaining is a process of calling properties and functions as an optional those values might be nil. If an optional variable has a value then it can be successfully call on the other hand it will return nil. e.g.

Var ans: Int?

Question # 10

Optional binding is a technique to safely unwrap optionals . To do this we use “if let ” statement

Var num: Int?

Num=9

If let num2 = num {

Print(num2)

}

Else

{

Print(“No value for num”)

}

Question #11

in function parameters are let constant we cannot change their values e.g

if I want to change the parameter value of the given function

func abc( num1: Int) -> Int{

num1 = num1\*2//////////////////// this line gives error

return num1

}

Abc(num)

If I want to change num1 values in the above function I have to use inout parameter like

func abc( num1: inout Int) -> Int{

num1 = num1\*2

}

Abc(num)

By writing inout we are dealing with reference type so that the variable which is passed from out side the function has overwritten values after calculation.

Question #12

Yes it is possible to give a default values to a parameter of a function e.g

**func** abc(num1: Int , cube1: Int = 3)-> Int

{

**var** result1 = 1

result1 = num1 \* cube1

**return** result1

}

abc(num1: 4)

in above example if we are not pass second parameter then it will take cube1 =3 .

Question #13

Forece unwraping unwrap an optional whether it contain a value or not. The only draw back of force unwraping is , when you assign a nil value to a variable then program will crashwhile unwraping it e.g.

**let** myname: String?

myname = **nil**

print(myname!)

Question #14

A method can read the value of the instant properties but cannot assign new value to them.

If we want a method to able to modify the value of the properties of its own instant we must declare the mutating keyword before func keyword in the structure definition. In classes we can change the properties values without mutating.

Struct namestd{

Var name: String

Mutating func sdtname(name1: String)

{

Name = name1

}

}

Question#15

Deinitializer is a builtin function which is used to release the memory space which is reserved by an instant of class. e.g.

Class subject{

Var name =” ”

Var author = “ ”

Deinit()

{

//////Code/////

}

}

}

Question#16

Protocol is defined with the keyword protocol followed by the name and list of properties and method between braces.

We are not allowed to assign the values to the properties and not writing any definition of its methods and functions.

Question #17

In object oriented programing code is implemented in side a class and then object are created from that class. if we need to create object with extra functionality we have to define a subclass that inherit the functionality from superclass and add some own functionality.

But in the protocol properties and methods which classes have common we declare them in the protocol and the definition is written in the classes.

Question #18

Struct Apple{

}

Func pick(apple: Apple?)

{

Guard let apple=apple else{

return

Print(“no apple found”)

}

Print(apple)

}

Question#19

**protocol** MainPro{

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

}

**protocol** SubPro1: MainPro{

**var** sub: String{**get**}

}

**protocol** SubPro2: SubPro1{

**var** num: Int { **get** }

}

**class** mainclass: SubPro2

{

**var** num: Int = 0

**var** sub: String = ""

**var** name: String = ""

}

Question # 20

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

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

first.append(contentsOf: second)

print(first