Values and Variables

1. Create 2 val types with x as 100 & y as 10 respectively and find the Multiplication and division
of both and store in some val as z and z1.

```
scala> val x:Int =100
x: Int = 100

scala> val y:Int =10
y: Int = 10

scala> val z:Int = x * y
z: Int = 1000

scala> val z1 = x / y
z1: Int = 10
```

2. Create a as 2000 and find the division of a by y created in step 1 and reassign a with the divided
result (200).

```
scala> var a=2000
a: Int = 2000

scala> a=a/y
a: Int = 200
```

3. Create a val type with x:Int=100, then assign the x to val y, but the datatype of y has to be String.

```
scala> val x:Int=100
x: Int = 100

scala> val y=x.toString
y: String = 100
```

4. Try only in REPL for now - Create a val type sc1 and assign sc into it and also try assigning sc1
defined as AnyRef/Any and check the type of the sc1 using getClass function.

```
scala> val sc1=sc
sc1: org.apache.spark.SparkContext = org.apache.spark.SparkContext@d8ab4c0

scala> val sc1:AnyRef = sc
```

```
sc1: AnyRef = org.apache.spark.SparkContext@d8ab4c0

scala> val sc1:Any = sc
sc1: Any = org.apache.spark.SparkContext@d8ab4c0

scala> sc1.getClass
res58: Class[_] = class org.apache.spark.SparkContext
```

Static definition and Dynamic inference

5. Create some var and val and prove static definition by re-assigning var with different data type
and dynamic inference by displaying the data type respectively.

```
scala> var t:AnyVal=100
t: AnyVal = 100

scala> t=20.3
t: AnyVal = 20.3
```

Conditional Structures

6. Write a program to find the greatest of 3 numbers

```
scala> def biggestThree(x:Int,y:Int,z:Int):Int={
    | if (x > y && x > z)
    |   return x
    | else if (y > x && y > z)
    |   return y
    | else
    |   return z
    | }
biggestThree: (x: Int, y: Int, z: Int)Int

scala> println(biggestThree(100,4999,599))
4999
```

7. Write a nested if then else to print the course fees of if the student choose bigdata then check if
bigdata then fees is 25000,
if spark then fees is 15000, if the student chooses datascience then check if machinelearning then
35000, if deep learning then 45000.

```
println("Enter the course name :")
```

```
val course = readLine()
val courseName = course match {
case "bigdata" => {
   println("Bigdata or Spark")
   val subcourse = readLine()
   if (subcourse == "bigdata")
     {
         println("The course fee is 25000")
         "Big Data"
     }
   else
     {
         println("The course fee is 15000")
         "Spark"
     }
}
case "datascience" => {
   println("ML or DL")
   val subcourse = readLine()
   if (subcourse == "ml")
     {
         println("The course fee is 35000")
         "Machine Learning"
     }
   else
     {
         println("The course fee is 45000")
         "Deep Learning"
     }
}
}
println("The course chosen : " + courseName)
```

8. Check whether the given string is palindrome or not (try to use some function like reverse). For
eg: val x="madam" then print as "palindrome" else "non palindrome".

```
scala> def palin(st:String):String={
    | if (st == st.reverse)
    |   "Palindrome"
    | else
    |   "Non Palindrome"
    | }
palin: (st: String)String
```

```
scala> println(palin("madam"))
Palindrome
```

9. Check whether the val x=100 is an integer or string. (try to use some functions like toString,
toUpperCase etc to execute this use case)

```
val x=100
if (x.isInstanceOf[Int])
println("x is an Integer")
else
println("x is a Non Integer")
```

Control Statements

10. Write a program using while or for loop to print even numbers and odd numbers between any
range of data as per your intention and also find the even and
odd values between 5 and 20 (even should be 6,8,10,12,14,16,18,20 and odd should be 5,7,9,11,13,15,17,19).

```
scala> val odd=scala.collection.mutable.ArrayBuffer[Int]()
odd: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer()

scala> val even=scala.collection.mutable.ArrayBuffer[Int]()
even: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer()

scala> for (x <- 5 until 20)
    | if (x%2 == 0)
    | even.append(x)

scala> for (x <- 5 until 20)
    | if (x%2 == 0)
    | even.append(x) else
    | odd.append(x)

scala> println(even)
ArrayBuffer(6, 8, 10, 12, 14, 16, 18, 6, 8, 10, 12, 14, 16, 18)

scala> println(odd)
ArrayBuffer(5, 7, 9, 11, 13, 15, 17, 19)
```

11. For loop to increment from 0 till 21 with the increment of 3, the result should be exactly 0,3,6,9,12,15,18

```
scala> for (i <- 0 until 21 by 3)
     | println(i)
```

12. Write a for or while loop to print the cube of 4, result should be 4*4*4=64 (think of using some
var type initiated outside the loop)

```
scala> import scala.util.control.Breaks._
import scala.util.control.Breaks._

scala> var x = 4
x: Int = 4

scala> for (i <- 0 to 10)
     | if (i == 4)
     |   print(i*i*i)
```

13. Write for/while loop for printing only the values in the range of 1 to 20 which are divisible by 4
(don't use by 4 in the for loop) rather use if condition to check the % of 4 for every element in the
loop achieve this.
Result should be exactly like this 4,8,12,16,20.

```
scala> for (i <- 1 to 20)
     | if (i%4 == 0)
     | println(i)
```

Methods

14. Write methods to make the above usecases (conditional and control statements from 7 to 13)
and upcoming usecases more generic by passing as parameters rather than hardcoding..

```
package org.inceptez.scalaprogram
import scala.io.StdIn._

object courseFee {
 def main(args:Array[String]):Unit=
 {
  println("Enter the course :")
  val course = readLine()

  println("You have chosen " + course)
  println("The course fee is " + courseFee(course))
 }

 def courseFee(courseName:String):Int={
  courseName match {
   case "bigdata" => {
       println("Enter BigData or Spark")
       val courseN1 = readLine()
       if (courseN1 == "bigdata")
        return 25000
       else
        return 15000
   }
   case "datascience" => {
       println("Enter ML or DL")
       val courseN = readLine()
       if (courseN == "ml")
        return 35000
       else
        return 45000
   }
  }
 }
}

scala> def forLoop(start:Int, end:Int){
   | for (i <- start to end)
   | println(i)
   | }
forLoop: (start: Int, end: Int)Unit

scala> forLoop(0,10)
0
1
2
3
4
5
6
7
8
9
10
```

15. Write a method to create a calculator accepts 3 arguments and return type of any, first 2 of
integer and 3rd one is String, based on the 3rd argument value as add/sub/div/mul perform either
addition or subraction or multiplication or division of values and return the result to the calling
environment. (for division think of using.toFloat or .toDouble conversion).

```
package org.inceptez.scalaprogram
import scala.io.StdIn._

object calculator {
  def main(args:Array[String]):Unit={
    println("Addition of two numbers " + calculators(10, 20, "add"))
    println("Subtraction of two numbers " + calculators(10, 20, "sub"))
    println("Multiplication of two numbers " + calculators(10, 20, "mul"))
    println("Division of two numbers " + calculators(30, 20, "div"))
  }

  def calculators(a:Int, b:Int, method:String):AnyVal={
    method match{
      case "add" => {a+b}
      case "mul" => {a*b}
      case "sub" => {a-b}
      case "div" => {a/b}
    }
  }
}
```

16. Try multiple return statements in a method and identify which one is really returning and what
are the returns are ignored.

```
package org.inceptez.scalaprogram

object greatest {

  def main(args:Array[String]):Unit=
  {
    println("The greatest number is " + greatestThree(210,140,100))
  }

  def greatestThree(a:Int, b:Int, c:Int):Int={
    if (a>b && a>c)
      return a
    else if (b>a && b>c)
```

```
      return b
    else
      return c
  }
}
```

17. Try creating a method with multiple return types.

```
package org.inceptez.scalaprogram
import scala.io.StdIn._

object calculator {
  def main(args:Array[String]):Unit={
     println("Addition of two numbers " + calculators(10.0, 20, "add"))
     println("Subtraction of two numbers " + calculators(10, 20.0, "sub"))
     println("Multiplication of two numbers " + calculators(10.0, 20, "mul"))
     println("Division of two numbers " + calculators(30, 20, "div"))
  }

  def calculators(a:Double, b:Double, method:String):AnyVal={
    method match{
      case "add" => {a+b}
      case "mul" => {a*b}
      case "sub" => {a-b}
      case "div" => {a/b}
    }
  }
}
```

Pattern matching

18. Write a program using case using pattern matching to find the datatype of a given value and
return either Float or string or Boolean or Char etc..

```
package org.inceptez.scalaprogram

object dataType {
  def main(args:Array[String]):Unit={
    println("The datatype is " + checkDataType(10))
    println("The datatype is " + checkDataType("Bala"))
    println("The datatype is " + checkDataType(10.0))
    println("The datatype is " + checkDataType(true))
    println("The datatype is " + checkDataType('G'))
  }
```

```
 def checkDataType(a:Any):String={
  a match
  {
   case int if (a.isInstanceOf[Int]) => {"Integer"}
   case string if (a.isInstanceOf[String]) => {"String"}
   case double if (a.isInstanceOf[Double]) => {"Double"}
   case boolean if (a.isInstanceOf[Boolean]) => {"Boolean"}
   case character if (a.isInstanceOf[Char]) => {"Character"}
   case _ => {"None"}
  }
 }
}
```

19. Create a method should accept 2 arguments and a return value metexception(numerator:Int,denominator:Int):Int={} , in the main block return the value of numerator/denominator

for eg. If you call metexception(10,2) the return should be 5 but if you call as metexception(10,0)
usually it throws exception, in case of exception we have to handle in the catch block where it
should call the same metexception with the argument passed as (10,1) so the result will be 10.

```
package org.inceptez.scalaprograms

object metException {
 def main(args:Array[String]):Unit=
 {
  println("Value is " + meetException(10,2))
  println("Value is " + meetException(10,1))
  println("Value is " + meetException(10,0))
 }
 def meetException(a:Int, b:Int):Int={
  try
   return a/b
  catch
  {
   case a:java.lang.ArithmeticException => {0}
  }
 }
}
```

Collections: Seq, Array, List, Map and Set
20. Create an array, list and prove mutability and immutability and non-resizable properties.

```
scala> val ar1=new Array[Int](3)
ar1: Array[Int] = Array(0, 0, 0)

scala> val ar2=new Array[String](3)
ar2: Array[String] = Array(null, null, null)

scala> ar1(0)=10

scala> ar1
res19: Array[Int] = Array(10, 0, 0)

scala> val lst = List(10,20,30)
lst: List[Int] = List(10, 20, 30)

scala> lst(0)
res20: Int = 10

scala> lst(0) = 5
<console>:29: error: value update is not a member of List[Int]
    lst(0) = 5
```

21. Create arraybuffer from scala.collection.mutable package and prove mutability and immutability and resizable properties.

```
scala> import scala.collection.mutable._
import scala.collection.mutable._

scala> val ab=ArrayBuffer(10,20,30)
ab: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 20, 30)

scala> ab
res24: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 20, 30)

scala> ab += 40
res25: ab.type = ArrayBuffer(10, 20, 30, 40)

scala> ab += (50,60)
res26: ab.type = ArrayBuffer(10, 20, 30, 40, 50, 60)

scala> ab ++= List(70,80,90)
res27: ab.type = ArrayBuffer(10, 20, 30, 40, 50, 60, 70, 80, 90)

scala> ab.append(100)
```

```
        scala> ab
        res29: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 20, 30, 40, 50, 60,
70, 80, 90, 100)

        scala> ab.remove(0)
        res30: Int = 10

        scala> ab
        res31: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(20, 30, 40, 50, 60, 70,
80, 90, 100)

        scala> ab -= 100
        res32: ab.type = ArrayBuffer(20, 30, 40, 50, 60, 70, 80, 90)

        scala> ab(0)=10

        scala> ab
        res34: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(10, 30, 40, 50, 60, 70,
80, 90)
```

22. Create a tuple of 4 fields and access the 2nd and 4th fields and store in another tuple.

```
        scala> val tup=(10,"Goms","BofA", 130000.0)
        tup: (Int, String, String, Double) = (10,Goms,BofA,130000.0)

        scala> tup._1
        res35: Int = 10

        scala> tup._2
        res36: String = Goms

        scala> tup._4
        res37: Double = 130000.0

        scala> val tup1=tup
        tup1: (Int, String, String, Double) = (10,Goms,BofA,130000.0)

        scala> tup1
        res38: (Int, String, String, Double) = (10,Goms,BofA,130000.0)

        scala> val tup2 = (tup._2, tup._4)
        tup2: (String, Double) = (Goms,130000.0)
```

23. Find the maximum value out of (2,3,1,5,4) elements in the array.

```
scala> val ar=Array(2,3,1,5,4)
ar: Array[Int] = Array(2, 3, 1, 5, 4)

scala> println(ar.max)
5
```

24. Find the max and min value of (2,3,1,5,4) elements in the array and store these 2 values in
another array.

```
scala> val ar=Array(2,3,1,5,4)
ar: Array[Int] = Array(2, 3, 1, 5, 4)

scala> println(ar.max)
5

scala> println(ar.min)
1

scala> val new_ar=new Array[Int](2)
new_ar: Array[Int] = Array(0, 0)

scala> new_ar(0) = ar.max

scala> new_ar(1) = ar.min

scala> new_ar
res45: Array[Int] = Array(5, 1)
```

25. Create a method to find the highest value in the given array if the array is non empty and print
it, you must pass array as an argument to the method.

```
package org.inceptez.scalaprograms

object maxArray {
  def main(args:Array[String]):Unit={
    val ar=Array(10,20,50,400,4,60)

    println("The maximum value in the given array is " + maxArrayValue(ar))

    val ar1=new Array[Int](3)
    println("The maximum value in the given array is " + maxArrayValue(ar1))

    val ar2: Array[Int]=Array()
    println("The maximum value in the given array is " + maxArrayValue(ar2))
```

```
 }

 def maxArrayValue(ar:Array[Int]):Int={
   if (ar.isEmpty)
     return -1
   else
     return ar.max
 }
}
```

26. Write a program to create an Int List with 5 different values using range and sum all the values

```
scala> val lst = List(Range(0,5))
lst: List[scala.collection.immutable.Range] = List(Range(0, 1, 2, 3, 4))

scala> val lst = List.range(1,10)
lst: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

27. Write a program to create string list to store the values of Spark,Scala,Python,Java,Hadoop and
count the number of elements in the List

```
scala> val stLst=List("Spark","Scala","Python","Java","Hadoop")
stLst: List[String] = List(Spark, Scala, Python, Java, Hadoop)

scala> println(stLst.length)
5
```

28. Write a program to store (China,Beijing),(India,New Delhi),(USA,Washington),(UK,London)
using Map

```
	scala> val map1=Map("China" -> "Beijing", "India" -> "New Delhi", "USA" -> 
"Washington", "UK" -> "London")
	map1: scala.collection.mutable.Map[String,String] = Map(China -> Beijing, India -> 
New Delhi, UK -> London, USA -> 		Washington)
```

29. Find the capital of India

```
scala> map1("India")
res49: String = New Delhi
```

30. Take only countries and store in an array and use foreach and println to print line by line of
elements.

```
scala> map1.values
res52: Iterable[String] = HashMap(Beijing, New Delhi, London, Washington)

scala> map1.keys
res53: Iterable[String] = Set(China, India, UK, USA)

scala> for ((x,y) <- map1)
    | println(x)
China
India
UK
USA

scala> var i=0
i: Int = 0

scala> for ((x,y) <- map1)
    | println(x)
China
India
UK
USA

scala> var country=new Array[String](4)
country: Array[String] = Array(null, null, null, null)

scala> for ((x,y) <- map1)
    | country(i) = x

scala> for ((x,y) <- map1)
    | {
    | country(i) = x
    | i = i+1
    | }

scala> country
res58: Array[String] = Array(China, India, UK, USA)

scala> map1.foreach(println)
(China,Beijing)
(India,New Delhi)
```

```
(UK,London)
(USA,Washington)
```

31. Take only countries and store in an set and use foreach and println to print line by line of
elements.

```
scala> import scala.collection.mutable._
import scala.collection.mutable._

scala> val conSet = Set[String]()
conSet: scala.collection.mutable.Set[String] = Set()

scala> map1
res62: scala.collection.mutable.Map[String,String] = Map(China -> Beijing, India ->
New Delhi, UK -> London, USA ->              Washington)

scala> for ((x,y) <- map1)
    | conSet.add(x)

scala> conSet
res64: scala.collection.mutable.Set[String] = Set(China, USA, UK, India)
```

32. Create a case class and apply the respective column name and datatype for the tuple created in
step 22.

<u>Print the number of words in a given sentence:</u>

Hello how are you doing

```
scala> var st = "Hello how are you doing"

scala> st.split(" ")
res69: Array[String] = Array(Hello, how, are, you, doing)

scala> st.split(" ").length
res70: Int = 5
```

<u>OOPS</u>

33. Create a package namely com.inceptez.datasecurity

```
package com.inceptez.datasecurity

object testOops {

}
```

34. Inside the above package, Create a class called mask and one more class called endecode

```
package com.inceptez.datasecurity

class mask {

}


package com.inceptez.datasecurity

class endecode {

}
```

35. Inside the class mask create a private val as addhash=100 and a method hashMask(str:String):Int={return the hashcode of str+addhash value}

```
package com.inceptez.datasecurity

class mask {
  private val addhash=100

  def hashMask(str:String):Int={
    return str.hashCode + addhash
    }
}
```

36. Inside the class endecode create a private val as prefixstr=”aix” and a method revEncode(str:String):String={return the prefixstr+reverse of str value}

```
package com.inceptez.datasecurity

class endecode {
 private val prefixstr="aix"

 def revEncode(str:String):String={
   return prefixstr+str.reverse
 }
}
```

37. Create a scala object namely singleobject, create a main method, create objects like
val objmask=new com.inceptez.datasecurity.mask;
val objencode=new com.inceptez.datasecurity.endecode;

```
package com.inceptez.datasecurity

object singleobject {
  def main(args:Array[String]):Unit={
    val objmask=new com.inceptez.datasecurity.mask;
    val objencode=new com.inceptez.datasecurity.endecode;
  }
}
```

38. Create an array with 3 names like Array("arun","ram kumar","yoga murthy"), loop the array elements, apply hashMask(name) for all 3 elements and println of the masked values.

```
package com.inceptez.datasecurity

object singleobject {
  def main(args:Array[String]):Unit={
    val objmask=new com.inceptez.datasecurity.mask;
    val objencode=new com.inceptez.datasecurity.endecode;

    var names = Array("arun","ram kumar","yoga murthy")

    for (x <- names)
      println(objmask.hashMask(x))
  }
}
```

39. Loop the array created in above step and apply the revEncode(name) for all 3 elements and println of the encoded values.

```
object singleobject {
  def main(args:Array[String]):Unit={
    val objmask=new com.inceptez.datasecurity.mask;
    val objencode=new com.inceptez.datasecurity.endecode;

    var names = Array("arun","ram kumar","yoga murthy")
    for (x <- names)
      println(objmask.hashMask(x))

    for (x <- names)
      println(objencode.revEncode(x))
  }
}
```

40. If possible create a decode function inside endecodeclass as revDecode and write a logic to decode the encoded string in step 39.

```
package com.inceptez.datasecurity

object singleobject {
  def main(args:Array[String]):Unit={
    val objmask=new com.inceptez.datasecurity.mask;
    val objencode=new com.inceptez.datasecurity.endecode;

    var names = Array("arun","ram kumar","yoga murthy")

    for (x <- names)
      println(objmask.hashMask(x))

    for (x <- names)
    {
      val rev = objencode.revEncode(x)
      println("The encoded value is :" + rev)

      println("The decoded value is :" + objencode.refDecode(rev))
    }
  }
}
```