1a

```
object SumCalculator {
  def sumOrTripleSum(a: Int, b: Int): Int = {
    if (a == b) {
      3 * (a + b)
    } else {
      a + b
    }
  }

  def main(args: Array[String]): Unit = {
    val num1 = 5
    val num2 = 6
    val result = sumOrTripleSum(num1, num2)
    println(s"The sum or triple sum of $num1 and $num2 is: $result")
  }
}
```

B

```
object Check2Num {
  def SumOrEqual(a: Int, b: Int): Boolean  = {
    if (((a == 22) || (b == 22)) || (a+b == 32)) {
      true
    } else {
      false
    }
  }

  def main(args: Array[String]): Unit = {
    val num1 = 10
    val num2 = 12
    val result = SumOrEqual(num1, num2)
    println(s"The sum = 32 or any one num is 22 for $num1 and $num2 is: $result")
  }
}
```

2a

```
object CharacterRemover {
  def removeCharacterAtPosition(inputString: String, position: Int): String = {
    if (position < 0 || position >= inputString.length) {
      return inputString
    }
```

```
    val (before, after) = inputString.splitAt(position)
    before + after.drop(1)
  }

  def main(args: Array[String]): Unit = {
    val inputString = "Sir M VIT"
    val position = 5
    val result = removeCharacterAtPosition(inputString, position)
    println(s"The string after removing character at position $position: $result")
  }
}
```

2B

```
object StringModifier {
  def addFirstFiveToFrontAndBack(inputString: String): String = {
    val firstFive = inputString.take(5)
    firstFive + inputString + firstFive
  }

  def main(args: Array[String]): Unit = {
    val inputString = "Sir MVIT"
    val result = addFirstFiveToFrontAndBack(inputString)
    println(s"The new string with the first five characters added at both front and back: $result")
  }
}
```

3a)

```
object MultiplicationTable {
  def main(args: Array[String]): Unit = {
    // Input the number for which you want to print the multiplication table
    print("Enter a number\t")
    val number = scala.io.StdIn.readInt()

    // Iterate through numbers from 1 to 10 and print their product with the input number
    for (i <- 1 to 10) {
      println(s"$number * $i = ${number * i}")
    }
  }
}
```

3b)

```
object LargestElement {
  def main(args: Array[String]): Unit = {
    println("Enter the size of the array:")
    val size = scala.io.StdIn.readInt()
    val arr = new Array[Int](size)
    println(s"Enter $size elements, one by one:")
    for (i <- 0 until size) {
      arr(i) = scala.io.StdIn.readInt()
    }
    println("Elements of the array:")
    for (elem <- arr) {
      println(elem)
```

```
    }
    println(s"Largest element in the array is: ${findLargest(arr)}")
  }

  def findLargest(arr: Array[Int]): Int = arr match {
    case Array(x) => x
    case Array(head, tail @ _*) =>{
      val maxTail = findLargest(tail.toArray)
      if (head > maxTail) head else maxTail
      }
    case _ => Int.MinValue
  }
}
```

4. a.

```
object productOfDigits {
  def productOfIndDigits(number: Int): Int = {
    // Convert the number to a string to iterate through its digits
    val digits = number.abs.toString.map(_.asDigit)
    println(digits)
    // Calculate the product of digits using foldLeft
    digits.foldLeft(1)(_ * _)
  }

  def main(args: Array[String]): Unit = {
    // Test the function
    print("Enter a number\t")
    val number = scala.io.StdIn.readInt()
    println(s"The product of digits in $number is: ${productOfIndDigits(number)}")
```

```
  }
}
```

B

```
object PerfectSquare {
  def main(args: Array[String]): Unit = {
    print("Enter a number\t")
    val number = scala.io.StdIn.readInt()
    println(s"$number is a perfect square: ${isPerfectSquare(number)}")

  }

  def isPerfectSquare(number: Int): Boolean = {
    // Take the square root of the number and check if it's an integer
    val squareRoot = math.sqrt(number)
    squareRoot.toInt * squareRoot.toInt == number
  }
}

class Person(var name: String, var age: Int)
 // extended class
 class Student(name: String, age: Int, var grade: Char) extends
Person(name, age) {   // Method to get the grade
  def getGrade: Char = grade
  // Method to set the grade
  def setGrade(newGrade: Char): Unit = { grade = newGrade  }
 }
```

5a

```
object Lab5Prg {
  def main(args: Array[String]): Unit = {
    print("Enter a Age\t")
    val age = scala.io.StdIn.readInt()
    print("Enter a Name\t")
    val Name = scala.io.StdIn.readLine()
    print("Enter a Grade\t")
    val Grade = scala.io.StdIn.readChar()
    val student = new Student(Name, age, Grade)
    // Get and print the initial grade
    println(s"Initial grade: ${student.getGrade}")
    // Set a new grade and print it
    print("Enter the Grade to Update\t")
    val UGrade = scala.io.StdIn.readChar()
    student.setGrade(UGrade)
    println(s"Updated grade from $Grade to :
            ${student.getGrade}")
  }
}
```

5b

```
class Triangle(val side1: Double, val side2: Double, val side3: Double) {
  // Method to check if the triangle is equilateral
  def isEquilateral: Boolean = {
    side1 == side2 && side2 == side3
  }
}
 object Lab5bPrg {
```

```
  def main(args: Array[String]): Unit = {
    print("Enter the three sides of a trianhgle\t")
    val s1 = scala.io.StdIn.readInt()
    val s2 = scala.io.StdIn.readInt()
    val s3 = scala.io.StdIn.readInt()
    val triangle1 = new Triangle(s1,s2,s3)
    println("Triangle :")
    println(s"Is equilateral:
          ${triangle1.isEquilateral}")
    // Should print true
  }
 }
```

6a

```
object Color extends Enumeration {
  type Color = Value
  val Red, Green, Blue, Yellow = Value
}

import Color._

object EnumeClass {
  def main(args: Array[String]): Unit = {
    val obj1Color: Color = Color.Red
```

```
    val obj2Color: Color = Color.Blue
    val obj3Color: Color = Color.Red

    println(s"Object 1 color: $obj1Color")
    println(s"Object 2 color: $obj2Color")
    println(s"Object 3 color: $obj3Color")
  }
}
```

6B

```
class ContactInfo(val name: String, val email: String, val address: String)

class Customer(val contactInfo: ContactInfo)

object EmpData {
  def main(args: Array[String]): Unit = {
    // Create a ContactInfo object
    print("Enter the name of the employee: ")
    val name = scala.io.StdIn.readLine()
    print("Enter the email address of the employee: ")
    val email = scala.io.StdIn.readLine()
    print("Enter the address of the employee: ")
    val address = scala.io.StdIn.readLine()

    val contact = new ContactInfo(name, email, address)
    val customer = new Customer(contact)
    println(s"Customer Name: ${customer.contactInfo.name}")
    println(s"Customer Email: ${customer.contactInfo.email}")
```

```
    println(s"Customer Address: ${customer.contactInfo.address}")
  }
}
```

7a

```
object SetAddDiff {
  def main(args: Array[String]): Unit = {


    println("Enter elements for Set 1 (separated by spaces):")
    val input1 = scala.io.StdIn.readLine()
    val set1 = input1.split(" ").map(_.toInt).toSet

    println("Enter elements for Set 2 (separated by spaces):")
    val input2 = scala.io.StdIn.readLine()
    val set2 = input2.split(" ").map(_.toInt).toSet

    val difference = set1.diff(set2)

    val intersection = set1.intersect(set2)



    println(s"Set 1: $set1")
    println(s"Set 2: $set2")
    println(s"Difference: $difference")
    println(s"Intersection: $intersection")
  }
}
```

7b

```
object SecondLarge {
  def main(args: Array[String]): Unit = {

    println("Enter elements for Set 1 (separated by spaces):")
    val input1 = scala.io.StdIn.readLine()
    val set = input1.split(" ").map(_.toInt).toSet

    val secondLargest = findSecondLargest(set)
    secondLargest match {
      case Some(value) => println(s"The second largest element in the set is: $value")
      case None => println("The set does not have enough elements to determine the second largest.")
    }
  }

  def findSecondLargest(set: Set[Int]): Option[Int] = {
    if (set.size < 2) {
      None // If set has less than two elements, return None
    } else {
      val sortedSet = set.toList.sorted // Convert set to a sorted list
      Some(sortedSet(sortedSet.length - 2)) // Return the second last element of the list
    }
  }
}
```

8. a. Write a Scala program to create a list in different ways. Note: Use Lisp style, Java style, Range list, Uniform list, Tabulate list

```
object TypesofLists {
  def main(args: Array[String]): Unit = {
   // Lisp style list
    val lispList = 1 :: 2 :: 3 :: Nil
   // Java style list
    val javaStyleList = List(1, 2, 3)
    // Range list
    val rangeList = (1 to 5).toList
    // Uniform list
    val uniformList = List.fill(5)(10)
    // Tabulate list
    val tabulateList = List.tabulate(5)(_ + 1)
   // Print lists
   println("Lisp style list: " + lispList)
   println("Java style list: " + javaStyleList)
   println("Range list: " + rangeList)
   println("Uniform list: " + uniformList)
   println("Tabulate list: " + tabulateList)
 }
}
```

8. b. Write a Scala program to flatten a given List of Lists, nested list structure.

```
object FlattenList {
 def main(args: Array[String]): Unit = {
   // Example list of lists
   val nestedList = List(
    List(1, 2, 3),
```

```
    List(4, 5),
    List(6, 7, 8, 9),
    List(10)
  )

  // Flatten the nested list
  val flattenedList = nestedList.flatten

  // Print the flattened list
  println(s"Flattened list: $flattenedList")
 }
}
```

10. a. Write a Scala program to swap the elements of a tuple Further print no swapping required if elements are same.

```
object SwapTupleElements {
 def main(args: Array[String]): Unit = {

  println("Enter first tuple (two values separated by a space):")
  val first = scala.io.StdIn.readLine()
  val Array(a, b) = first.split(" ").map(_.toInt)

  val tuple1: (Int, Int) = (a, b)
  println("Tuple: " + tuple1)

  val swappedTuple1 = swapTuple(tuple1)
  println("Tuple 1 after swapping: " + swappedTuple1)
 }
 def swapTuple(tuple: (Int, Int)): (Int, Int) = {
```

```
    tuple match {
      case (a, b) if a != b => (b, a)
      case _ => tuple
    }
  }
}
```

B

```
object NotUnique {
  def main(args: Array[String]): Unit = {
    val tuple = (1, 2, 2, 3, 4, 4, 4, 5)
    val nonUnique = findNonUniqueElements(tuple)
    println("Non-unique elements in the tuple: " + nonUnique)
  }
  def findNonUniqueElements(tuple: Product): Set[Any] = {
    tuple.productIterator
      .toList
      .groupBy(identity)
      .collect {
        case (elem, occurrences) if occurrences.size > 1 => elem
      }
      .toSet
  }
}
```

9a

```
object AddEleNTimes {
  def main(args: Array[String]): Unit = {
```

```
   println("Enter elements for List 1 (separated by spaces):")
   val input1 = scala.io.StdIn.readLine()
   val list = input1.split(" ").map(_.toInt).toList

   print("Enter How many times to be repeated\t")
   val n = scala.io.StdIn.readInt()
   val newList = addNTimes(list, n)

   println(s"Original List: $list")
   println(s"Each element added $n times: $newList")
  }

  def addNTimes(list: List[Int], n: Int): List[Int] =
{
   list.flatMap(e => List.fill(n)(e))
  }
}
```

9b

```
object SplitList {
  def main(args: Array[String]): Unit = {

    println("Enter elements for List 1 (separated by spaces):")
   val input1 = scala.io.StdIn.readLine()
   val list = input1.split(" ").map(_.toInt).toList
```

```
    val (firstHalf, secondHalf) = splitList(list)

    println("First Half: " + firstHalf)
    println("Second Half: " + secondHalf)
  }

  def splitList[A](list: List[A]): (List[A], List[A]) = {
    val middleIndex = list.length / 2
    list.splitAt(middleIndex)
  }
}
```