Assignment

Submitted to

DEVI AHILYA VISHWAVIDHYALYA INDORE

In Partial Fulfillment of the Requirements for the Degree of

MASTER OF TECHNOLOGY

in

BIG DATA ANALYTIC

by

Er. Manish Kumar Singh
DS7B-2208

Under the Supervision of

Mr. Makhan Khumbhkar
Assistant Professor

SCHOOL OF DATA SCIENCE AND FORECASTING

2022-23

scala_Assignment

April 7, 2023

Assignment 1

Write Scala functions to solve the following problems.

1. The temperature is 35C; convert this temperature into Fahrenheit. ºF =ºC * 1.8000 + 32.00

[ ]:
```
object Fahrenheit extends App{

  def convert(celsius:Double)=celsius*1.8000+32.00;
  println("fahrenheit value for "+ 35 +" celsius is "+convert(35));
}
```

```
Intitializing Scala interpreter …
```

2.The volume of a sphere with radius r is 4/3 r3. What is the volume of a sphere with radius 5?

[ ]:
```
object VolumeSphere extends App{

      def volume(r:Double)=4.0/3.0*math.Pi*r*r*r;

      println(volume(5))

}
```

3. Suppose the cover price of a book is Rs. 24.95, but bookstores get a 40% discount.Shipping costs Rs. 3 for the first 50 copy and 75 cents for each additional copy. What is the total wholesale cost for 60 copies?

[ ]:
```
object BookStore extends App{
  def totalCost(copies:Double)=24.95*copies-24.95*0.4*copies+3.0*50.0+0.75*10.0;
  println(totalCost(60))
}
```

Assignment 2

Write Scala functions to solve the following problems.

1. Company XYZ & Co. pays all its employees Rs.150 per normal working hour and Rs. 75 per OT hour. A typical employee works 40 (normal) and 20(OT) hours per week has to pay 10% tax. Develop a functional program that determines the take home salary of an employee from the number of working hours and OT hours given

[ ]:
```
import scala.io.StdIn._

object TakeHomeSalary extends App{

        def wage(normalHour:Int)=normalHour*150

        def ot(otHour:Int)=otHour*75

        def tax(income:Int)=income*0.1

        def income(normalHour:Int,otHour:Int)=wage(normalHour)+ot(otHour)


        printf("Enter Normal Hours :")
        var normalHour=readInt()
        printf("Enter OT Hours :")
        var otHour=readInt()

        printf("Take home salary = %.2f␣
 ↪\n",income(normalHour,otHour)-tax(income(normalHour,otHour)))
}
```

2. Imagine the owner of a movie theater who has complete freedom in setting ticket prices. The more he charges, the fewer the people who can afford tickets. In a recent experiment the owner determined a precise relationship between the price of a ticket and average attendance. At a price of Rs 15.00 per ticket, 120 people attend a performance. Decreasing the price by 5 Rupees increases attendance by 20 and increasing the price by 5 Rupees decreases attendance by 20. Unfortunately, the increased attendance also comes at an increased cost. Every performance costs the owner Rs.500. Each attendee costs another 3 Rupees. The owner would like to know the exact relationship between profit and ticket price so that he can determine the price at which he can make the highest profit. Implement a functional program to find out the best ticket price.

[ ]:
```
import scala.io.StdIn._

object BestTicketPrice extends App{

  def attendee(price:Double)=120+(15-price)/5*20

  def revenue(price:Double)=price*attendee(price)

  def cost(price:Double)=500+3*attendee(price)

  def profit(price:Double)=revenue(price)-cost(price)

  printf("Enter the Ticket Price :")
  var tprice=readDouble()
  printf("Profit for Given Ticket Price : %.2f \n ",profit(tprice))

}
```

Assignment 3

1. Can you write a recursive function prime(n) that returns true for a prime number and false for the other? For example prime(5) should return true and prime(8) should return false.

[ ]:
```
import scala.io.StdIn._

object PrimeNumber extends App{

  def gcd(a:Int,b:Int):Int=b match{
    case 0=>a
    case b if(b>a) =>gcd(b,a)
    case _ =>gcd(b,a%b)
  }

  def isPrime(n:Int,i:Int):Boolean= n match{
    case n if n==i => true

    case n if gcd(n,i)>1 => false

    case x  =>isPrime(n,i+1)

   }


  print("Enter a Number : ")
  var n=readInt()

  println(isPrime(n,2))

}
```

2. Can you write a recursive function primeSeq(n) that prints all prime number which less than n? For example prime(10) should print 2,3,5, and 7 on the terminal.

[ ]:
```
import scala.io.StdIn._
import scala.util.control.Breaks._

object PrimeNumberSeq extends App{

  def gcd(a:Int,b:Int):Int=b match{
    case 0=>a
    case b if(b>a) =>gcd(b,a)
    case _ =>gcd(b,a%b)
  }
  def isPrime(n:Int,i:Int):Boolean= n match{
    case n if n==i => true

    case n if gcd(n,i)>1 => false

    case x  =>isPrime(n,i+1)

   }

  def printSeq(n:Int,i:Int){
     if(n==i) {
        break
     }

     if(isPrime(i,2)){
       println(i)
     }

     printSeq(n,i+1)

  }
  print("Enter a Number : ")
  var n=readInt()

  printSeq(n,2)

}
```

3. Can you write a recursive function returns the addition of numbers from1 to n? For example sum(5) should print 15

[ ]:
```
import scala.io.StdIn._

object Addition extends App{

  def addition(n:Int):Int={

     if(n==0)
        0
     else
       n+addition(n-1)

  }
  print("Enter a Number : ")
  var n=readInt()

  println(addition(n))

}
```

4. Can you write a recursive function to determine even and odd numbers?

[ ]:
```
import scala.io.StdIn._

object AdditionEven extends App{

  def evenAddition(n:Int):Int={
     if(n%2==0){
        if(n==0)
           0
        else
           n+evenAddition(n-1)
     }
     else{
        evenAddition(n-1)
     }

  }



  print("Enter a Number : ")
  var n=readInt()

  println(evenAddition(n))

}
```

5. Can you write a recursive function to get the addition of all even numbers less than given n.

[ ]:
```
import scala.io.StdIn._

object AdditionEven extends App{

  def evenAddition(n:Int):Int={
     if(n%2==0){
        if(n==0)
           0
        else
           n+evenAddition(n-1)
     }
```

```
    else{
       evenAddition(n-1)
    }

  }



  print("Enter a Number : ")
  var n=readInt()

  println(evenAddition(n))

}
```

6. The Fibonacci Sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . Each subsequent number is the sum of the previous two. Write a recursive function print fist n Fibonacci numbers for given n.

[ ]:
```
import scala.io.StdIn._
import scala.util.control.Breaks.break

object Fibo extends App{

  def fib(n:Int):Int={

     if(n<=1)
        n
     else
        fib(n-1)+fib(n-2)

  }

  def printFibo(n:Int,i:Int){

      if(n==i)
          break
       else
         println(fib(i))
      printFibo(n,i+1)

}

  print("Enter a Number : ")
  var n=readInt()

  printFibo(n,0)
}
```

Assignment 4

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals. 1. Implement Encryption and Decryption functions of Caesar cipher. 2. Then implement a Cipher function which take Encryption and Decryption functions to process the data.

[ ]:
```
import scala.io.StdIn._

object CaesarCipher {

    def main(args: Array[String]){

     val alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

     //READ WORD
     print("Enter the Word >")
     var word=readLine()
     //Shifted key ?
     print("Enter Shifted key >")
     var shift=readInt()
     //ASK E OR D
     print("Enter E-Encrypt or D-Decrypy >")
     val flag:Char=readChar().toUpper

     val E =(c:Char,shift:Int,a:String)=> a((a.indexOf(c.toUpper)+shift)%26)

     val D =(c:Char,shift:Int,a:String)=> a((a.indexOf(c.toUpper)-shift)%26)


      val cipher=(algo:(Char,Int,String)=>Char,word:String,a:String,shift:
 ↪Int)=> word.map(algo(_,shift,a));

      //CALL

      val c= if(flag=='E') cipher(E,word,alphabet,shift) else␣
 ↪cipher(D,word,alphabet,shift)
      println(c);


    }

}
```

Assignment 5

1. Implement a Data Structure for Rational Number and create a method neg to class Rational that is used like this: x.neg // evaluates to –x

[ ]:
```
class Rational(x:Int,y:Int) {
   var numer=x;
   var denom=y;

   def neg()= new Rational(-numer,denom);
}

object First {
   def main(args: Array[String]) {
      val x:Rational=new Rational(3,4);
      //1. neg
       var r:Rational=x.neg;
       println(r.numer+"/"+r.denom);


   }
}
```

2. Create a method sub to subtract two rational numbers and find an answer x-y-z where x=3/4, y=5/8, z=2/7.

[ ]:
```
class Rational(x:Int,y:Int) {
   var numer=x;
   var denom=y;


   def sub(r:Rational)={
       new Rational(numer*r.denom - r.numer*denom,denom * r.denom);
   }

 }




object Second{
   def main(args: Array[String]) {

       val x:Rational=new Rational(3,4);
       val y:Rational=new Rational(5,8);
       val z:Rational=new Rational(2,7);

       var k:Rational=x.sub(y);
       var r:Rational=k.sub(z);
```

```
        println(r.numer+"/"+r.denom);


    }
}
```

3. Implement a Data Structure for Account and create a method transfer which transfer the money from this account to a given account.

[ ]:
```
class Account(id:String,n: Int, b: Double) {
     val nic:String=id;
     val acnumber:Int=n;
     var balance:Double=b;

     def transfer(acc:Account,amount:Double)= {
         this.balance=this.balance-amount;
         acc.balance=acc.balance+amount;
     }

}


object Third{
   def main(args: Array[String]) {

       val a:Account=new Account("981234567V",10012,10000.00);
       val b:Account=new Account("951234567V",10023,12000.00);

       a.transfer(b,2000.00);

       println("Balance of b after transfered :"+b.balance);


   }
}
```

4. A Bank defines as List of Accounts. So implement the following functions: 4.1 List of Accounts with negative balances 4.2 Calculate the sum of all account balances 4.3 Calculate the final balances of all accounts after apply the interest function as fallows: If balance is positive, deposit interest is .05 and if balance is negative, overdraft interest is .1

[ ]:
```
class Account(id:String, n:Int, b:Double){
    val nic:String = id
    val accNum:Int = n
    var balance:Double = b

    override def toString= "["+nic+":"+accNum+":"+balance+"]"
}

object Fourth{

 def main(args: Array[String]) {

        //List of acc


       val a:Account=new Account("981234567V",10012,-500.00);
       val b:Account=new Account("951234567V",10023,1200.00);
        var bank:List[Account]=List(a,b);

       //overdraft acc
       val overdraft = bank.filter(x=>x.balance<0)
       println("Over Draft Accounts : "+overdraft);


        // Sum
       var total = bank.map(x=>x.balance).reduce((x,y) => x+y)
       println("Total Balance = " +total)

       //interest
       bank.map(x=> if(x.balance>0) x.balance=x.balance*1.05 else x.balance=x.
↪balance*1.1)
       println("With Interest :"+bank);}
 }
```

Assignment 6

Implement a case class Point(x,y) and create following methods: 1. add(+) should add two given points 2. move should move a point by a given distance dx and dy 3. distance should return the distance between two given points 4. invert should switch the x and y coordinates.

[ ]:
```
import scala.math._

object CaseClass extends App {

    val p1=Point(4,5);
    val p2=Point(2,3);

    //call +
    println(p1+" + "+p2+"= "+ p1.+(p2)); //(6,8)

    //move the point by (1,1)
    println(p1+" Move by (1,1) :"+ p1.move(1,1));

    //distance between p1 and p2
    println("Distance between "+p1+" and "+p2+" = " + p1.distance(p2));

    //switch x and y
    println("Invert of "+p1+" = "+ p1.invert()); //(5,4)

}

case class Point(a:Double,b:Double){
    def x:Double=a;
    def y:Double=b;

    ////+ Operator
    def +(point:Point)=Point(this.x+point.x,this.y+point.y);

    //move the point
    def move(dx:Int,dy:Int)=Point(this.x+dx,this.y+dy);

    //Distance between 2 given points
    def distance(p1:Point):Double={
           val y2:Double=pow(p1.y-this.y,2);
          val x2:Double=pow(p1.x-this.x,2);
          return sqrt(y2+x2);
    }

    //switch x and y
    def invert()=Point(this.y,this.x);

}
```

List based assignment

Assignment 1

Every Student has some marks associated with it. Student details contains its id and name. And for Marks, there are subjectId, studentId and number of marks a student scored.

Following are the requirements which is required to gain from above scenario (i.e. Student and marks)

1. Input:- (subjectId, percentage, pass/fail)

- Output:- for input pass, evaluate that how much students(id, name) are passed in the inputted subjectId for input fail, evaluate that how much students(id, name) are failed in the inputted subjectId
- Note:- percentage is the input which defines the minimum passing criteria
- e.g. Pass count: 15 Fail count: 10

2. Input:- (subjectId, count, top/bottom)

- Output:- based on the last input(top/bottom), output the students details who have scored max/min in that subjectId

- e.g.
- input: 1 5 top
- output: Kunal 85 Himanshu 84 Geetika 83 Anmol 82 Mahesh 81

3. Input:- (top/bottom, count)

- OutPut:- Overall top/least scorer based on all the subjects score, fetch students name count-input defines that how much students name are to be printed on console
- e.g. input: top 2
- output: Himanshu 75% Geetika 74%

4. Input:- (percentage, good_scholarship, normal_or_no_scholarship)

- Output:- two groups of students with the amount of scholarship
- e.g. input: 85% 2000 500
- output: Kunal 2000 Himanshu 500 Geetika 2000 Mahesh 500

5. Input:- (pass/fail, percentage) count and print the number of students and all names who are passed/fail, Pass or fail would be decided by percentage input field.

- e.g. input: fail 30
- output: Kunal 28% Himanshu 29%

6. Find the student(s) who have scored 95% or above and print its details.

- input: 95%
- output: Kunal 95% Himanshu 96% Geetika 97%

7. For every student, find its marks in detail (just like detailed Report card of a student.)

- Note:- must use groupBy method of List
- input: reportcard
- output: Kunal 75 70 80 75 75% Himanshu 74 70 81 75 75% Geetika 70 70 85 75 75%

[ ]:
```
/**
Assignment1
*/

case class Student(id: Long, name: String)

case class Marks(subjectId: Int,studentId: Long, marksObtained: Float)

class MarksManipulation()
{

    def pass_fail_count(sub_id:Long,percent:Float,pass_fail:String,source_marks:
 ↪List[List[List[Marks]]]):String=

    {

        val fetch_list=source_marks flatMap(_ flatMap(_␣
 ↪map(x=>check_count(x,sub_id,percent))))
```

```
      val pass_list=fetch_list.filter{_==1}
      val fail_list=fetch_list.filter{_==0}
      val passcount=pass_list.size
      val failcount=fail_list.size
      if(pass_fail.toLowerCase=="pass") s"Passcount = $passcount" else␣
↪s"Failcount= $failcount"
   }
   def check_count(element:Marks,id:Long,percent:Float):Int=

   {

      if(id==element.subjectId && element.marksObtained>percent) 1
      else if(id==element.subjectId && element.marksObtained<percent) 0 else␣
↪-1
   }

   def top_bottom(sub_id:Long,count:Int,topBottom:String,source_marks:
↪List[List[List[Marks]]],source_student:List[Student]):String=
   {
      val fetch_list=source_marks flatMap(_ flatMap(_␣
↪map(x=>support_top_bottom(x,sub_id))))
      val intended_list=fetch_list. filter{_.marksObtained!=0}
      val sortedList=intended_list.sortWith(_.marksObtained > _.marksObtained)
      if(topBottom.toLowerCase=="top")
      get_names(count,sortedList,source_student)
              else
              {
                      val revSortedList=sortedList.reverse
                      get_names(count,revSortedList,source_student)
              }
      }
      def get_names(count:Int,sortedList:List[Marks],source_student:
↪List[Student]):String=
      {
              for(check1<-0 to count)
              {
                      for(check2<-0 until source_student.size )
                      {
                              if(sortedList(check1).
↪studentId==source_student(check2).id)
                              {
                                      println(source_student(check2).name+"␣
↪"+sortedList(check1).marksObtained+"%")
                              }
                      }
```

```
                }
                "These are the names!!"
        }
        def support_top_bottom(element:Marks,id:Long):Marks=
        {
                if(element.subjectId==id)element else Marks(0,0,0)
        }
        def overallTopBottom(count:Int,topBottom:String,source_marks:
↪List[List[List[Marks]]],source_student:List[Student]):String=
        {

                val␣
↪namesWithMarks=namesWithPercentage(source_marks,source_student)
                val sort=namesWithMarks sortBy(_._1)
                if(topBottom.toLowerCase =="bottom")
                        printNames(count,sort)
                else
                {
                        val revList=sort.reverse
                        val result=printNames(count,revList)
                        result
                }
        }
        def sortAccordingToStudentId(source:List[Marks]):List[Marks]=
                {source.sortWith(_.studentId < _.studentId)}
        def folding_function(source:List[Marks]):Double=
        {
                val extract=for{
                        check<-0 until source.size
                } yield source(check).marksObtained
                val listExtract=extract.toList
                val result=listExtract.foldLeft(0.
↪0){(first,second)=>first+second}
                result
        }
        def only_names(students:List[Student]):List[String]=
        {
                val result=for{check<-0 until students.size}yield␣
↪students(check).name
                result.toList
        }
        def printNames(count:Int,marksNames:List[(Double,String)]):String=
        {
                val drop_list= marksNames.slice(0,count)
                val (marks,names)=drop_list.unzip
                for(check<-0 until names.size)
                        println(names(check)+"  "+marks(check)+"%")
                "These are the names!"

        }
        def scholarship(percent:Float,amount:Long,source_marks:
↪List[List[List[Marks]]],source_student:List[Student]):String=
        {

                val␣
↪namesWithPercent=namesWithPercentage(source_marks,source_student)
                val sort=namesWithPercent sortBy(_._1)
                val finalNames=IsScholarship(percent,sort)
                finalNames.map(x=>println(x+" "+amount))
                "Well done guyzz!!"

        }
        def IsScholarship(percent:Float,PercenageNames:List[(Double,String)]):
↪List[String]=
        {
                val(percentage,names)=PercenageNames.unzip
                val isAllowed=for{
                                check<-0 until percentage.size
                        }yield if(percentage(check)>=percent) check else 0
                val isAllowed_List=isAllowed.toList
                val filteredList=isAllowed_List.filter{_>0}
                val finalNames=for{
                                check<-0 until filteredList.size
                        }yield names(filteredList(check))
                finalNames.toList
        }
        def passFailNames(passFail:String,percent:Float,source_marks:
↪List[List[List[Marks]]],source_student:List[Student]):String=
        {

                val␣
↪namesWithPercent=namesWithPercentage(source_marks,source_student)
                val(percentage,name)=namesWithPercent.unzip
                val indexOfNames=for{
                                check<-0 until percentage.size
                        }yield if(percentage(check)>=percent)check else -1
                val indexOfNamesList=indexOfNames .toList
                if(passFail.toLowerCase=="pass")
                {
                        val ispass=indexOfNamesList.filter(_>=0)
                        val passNames=for{
                                        check<- 0 until ispass.size
                                }yield name(ispass(check))
                        for(check<-0 until passNames.size)
                        {
                                println(passNames(check)+"␣
↪"+percentage(ispass(check))+"%")
                        }
                        "PASS STUDENT NAMES!!"
                }
                else
                {
                        val indexOfFail=for{
                                check<-0 until percentage.size
                        }yield if(percentage(check)<percent)check else -1
                        val indexOfFailList=indexOfFail.toList
                        val isFail=indexOfFailList.filter(_>=0)
                        val failName=for{
                                        check<- 0 until isFail.size
                                }yield name(isFail(check))
                        for(check<-0 until failName.size)
                        {
                                println(failName(check)+"␣
↪"+percentage(isFail(check))+"%")
                        }
                        "FAIL STUDENT NAMES!!"
                }
        }
        def namesWithPercentage(source_marks:
↪List[List[List[Marks]]],source_student:List[Student]):List[(Double,String)]=
        {
                val fetch_list=source_marks flatMap(_ flatMap(_ map(x=>x)))
                val sort_list=sortAccordingToStudentId(fetch_list)
                val aggregate=for{
                                check<-0 to fetch_list.size/5-1
                        }yield if(check==0){folding_function(sort_list.
↪slice(0,5))} else {folding_function(sort_list.slice((check*5),(check+1)*5))}
                val aggregate_list=aggregate.toList
                val percentList=aggregate_list.map(x=>x/5)
                val names=only_names(source_student)
                percentList zip(names)
        }
        def topStudents(percent:Float,source_marks:
↪List[List[List[Marks]]],source_student:List[Student]):String=
        {
                val␣
↪namesWithPercent=namesWithPercentage(source_marks,source_student)
                val(percentage,name)=namesWithPercent.unzip
                val indexOfBrilliance=for{
                                check<-0 until percentage.size
                        }yield if(percentage(check)>=percent)check else -1
                val indexOfBrillianceList=indexOfBrilliance.toList.filter{_>0}
                val rareName=for{
                                            check<- 0 until indexOfBrillianceList.
↪size
                                }yield name(indexOfBrillianceList(check))
                for(check<-0 until indexOfBrillianceList.size)
                {
                        println(rareName(check)+"␣
↪"+percentage(indexOfBrillianceList(check))+"%")
                }
                "The Gems!!"
        }
        /*def report(reportcard:String,source_marks:
↪List[List[List[Marks]]],source_student:List[Student])=
        {
                val fetch_list=source_marks flatMap(_ flatMap(_ map(x=>x)))
                val stud_1_Name=

        }*/
}
object DataProvider
{

        def main(args: Array[String])
        {

                val student_1 = Student(1,"Nitin Aggarwal")

                val student_2 = Student(2,"Vandana Yadav")

                val student_3= Student(3,"Isha Jain")

                val student_4 = Student(4,"Kavit Pandey")

                val student_5 = Student(5,"Goldy Gupta")

                val student_6 = Student(6,"Manoj Tomar")

                val student_7 = Student(7,"Ankur Thakur")

                val student_8 = Student(8,"Avreen Kaur")

                val student_9 = Student(9,"Pooja Saini")

                val student_10 = Student(10,"Tarun Sinha")
                val maths_1=Marks(1,1,86)

                val maths_2=Marks(1,2,85)

                val maths_3=Marks(1,3,96)

                val maths_4=Marks(1,4,76)

                val maths_5=Marks(1,5,70)

                val maths_6=Marks(1,6,69)

                val maths_7=Marks(1,7,30)

                val maths_8=Marks(1,8,38)

                val maths_9=Marks(1,9,28)

                val maths_10=Marks(1,10,39)


                val science_1=Marks(2,1,88)

                val science_2=Marks(2,2,83)

                val science_3=Marks(2,3,95)

                val science_4=Marks(2,4,70)

                val science_5=Marks(2,5,65)

                val science_6=Marks(2,6,67)

                val science_7=Marks(2,7,37)

                val science_8=Marks(2,8,55)

                val science_9=Marks(2,9,34)

                val science_10=Marks(2,10,25)


                val hindi_1=Marks(3,1,85)
                val hindi_2=Marks(3,2,86)

                val hindi_3=Marks(3,3,94)

                val hindi_4=Marks(3,4,73)

                val hindi_5=Marks(3,5,68)

                val hindi_6=Marks(3,6,62)

                val hindi_7=Marks(3,7,40)

                val hindi_8=Marks(3,8,35)

                val hindi_9=Marks(3,9,44)

                val hindi_10=Marks(3,10,30)


                val Eng_1=Marks(4,1,86)

                val Eng_2=Marks(4,2,85)

                val Eng_3=Marks(4,3,97)

                val Eng_4=Marks(4,4,76)

                val Eng_5=Marks(4,5,70)

                val Eng_6=Marks(4,6,69)

                val Eng_7=Marks(4,7,30)

                val Eng_8=Marks(4,8,38)

                val Eng_9=Marks(4,9,28)

                val Eng_10=Marks(4,10,39)


                val comp_1=Marks(5,1,88)

                val comp_2=Marks(5,2,83)
```

```
                val comp_3=Marks(5,3,99)

                val comp_4=Marks(5,4,70)

                val comp_5=Marks(5,5,65)

                val comp_6=Marks(5,6,67)

                val comp_7=Marks(5,7,37)

                val comp_8=Marks(5,8,55)

                val comp_9=Marks(5,9,34)

                val comp_10=Marks(5,10,25)



                val␣
↪student_list=List(student_1,student_2,student_3,student_4,student_5,student_6,student_7,stu

                val Marks_stud_1=List(maths_1,science_1,hindi_1,Eng_1,comp_1)

                val Marks_stud_2=List(maths_2,science_2,hindi_2,Eng_2,comp_2)

                val Marks_stud_3=List(maths_3,science_3,hindi_3,Eng_3,comp_3)

                val Marks_stud_4=List(maths_4,science_4,hindi_4,Eng_4,comp_4)

                val Marks_stud_5=List(maths_5,science_5,hindi_5,Eng_5,comp_5)

                val Marks_stud_6=List(maths_6,science_6,hindi_6,Eng_6,comp_6)

                val Marks_stud_7=List(maths_7,science_7,hindi_7,Eng_7,comp_7)

                val Marks_stud_8=List(maths_8,science_8,hindi_8,Eng_8,comp_8)

                val Marks_stud_9=List(maths_9,science_9,hindi_9,Eng_9,comp_9)

                val␣
↪Marks_stud_10=List(maths_10,science_10,hindi_10,Eng_10,comp_10)



                val␣
↪marks_list=List(List(Marks_stud_1),List(Marks_stud_2),List(Marks_stud_3),List(Marks_stud_4)
                val obj=new MarksManipulation()
                val passFail=obj.pass_fail_count(1,40,"pass",marks_list)




        println(passFail)
        val topBottomSub=obj.top_bottom(5,4,"bottom",marks_list,student_list)
        println(topBottomSub)
        val topBottomOverall=obj.
↪overallTopBottom(3,"top",marks_list,student_list)
        println(topBottomOverall)
        val names=obj.scholarship(85,2000,marks_list,student_list)
        println(names)
        val pass_Fail=obj.passFailNames("fail",40,marks_list,student_list)
        println(pass_Fail)
        val top=obj.topStudents(95,marks_list,student_list)
        println(top)
        //obj.report("reportcard",marks_list,student_list)
    }
```

Developer Notes: There would be two case classes - Student(id: Long, name: String) - Marks(subjectId: Int,studentId: Long, marksObtained: float) In order to fill data in those case classes, either take inputs from a file, or take static inputs. But there must be atleast 5 subjects, and atleast 10 students. - e.g. List(Student(1, "Kunal"), Student(2, "Himanshu"), Student(3, "Geetika") ….) List(Marks(1, 1, 95), Marks(2, 1, 75), …) So basically here Kunal has marks 95 and 75 for the paper 1 and 2 respectively.

Assignment 2

- Find the last element of list with its index value(dont use inbuilt methods to extract last element directly)

[ ]:
```
/**
 Find the last element of list with its index value(dont use inbuilt methods to␣
 ↪extract last element directly)
*/

class Elements
{

def lastElement(list:List[Int],index:Int):(Int,Int)= // using tail recursion
{
val element=list.head

if((list.tail).isEmpty) (element,index) else lastElement(list.tail,index+1)
}
}

object LastElement extends App {
val list = List(1,2,3,4)
val obj1 = new Elements
val result=(obj1.lastElement(list,0))
println("last elemnt="+result._1)
println("index is ="+result._2)


}
```

- print the table of each element in the List

[ ]:
```
/**
 print the table of each element in the List
*/

object TableDemo extends App {
  val list = List(1,2,3,4)
  for(i<-list)
  {
    for(j<-1 to 10)
    {
      print(i*j+" ")

    }
    println("")

  }

}
```

- aggregate the contents of two lists of same size into a single list List(1,2) and List("a", "b") results List(List(1, "a"), List(2, "b"))

[ ]:
```
/**
aggregate the contents of two lists of same size into a single list
List(1,2) and List("a", "b") results List(List(1, "a"), List(2, "b"))
*/

object Aggregate extends App
{
    val lst1 = List(1,2,3)
    val lst2 = List("a","b","c")
    val lst = lst1 zip lst2
    val list = lst.map(x => List(x._1, x._2))
    println(list)
}
```

- find sum and multiplication of the list (dont use inbuilt methods)

[ ]:
```
/**
find sum and multiplication of the list (dont use inbuilt methods)
*/

class SumMul
{
    var lst3 : List[Int] = List()
    var lst4 : List[Int] = List()
    def sum(lst1:List[Int], lst2:List[Int])={
        var l1 = for{
            index <- 0 to lst1.length - 1
            lst3 = lst1(index) + lst2(index)
        }yield lst3
        println(l1)
    }
    def mul(lst1:List[Int], lst2:List[Int])={
        var l2 = for{
            index <- 0 to lst1.length - 1
            lst4 = lst1(index) * lst2(index)
        }yield lst4
        println(l2)
    }
}
object SumMulList extends App
{
    val lst1 = List(1,2,3)
    val lst2 = List(2,3,4)
    val lst = new SumMul
    lst.sum(lst1,lst2)
    lst.mul(lst1,lst2)
}
```

- apply quicksort and mergesort on the Lists

[ ]:
```
/**
apply mergesort on the Lists
*/

object MergeSort1 extends App
{
def mergeSort(lst: List[Int]): List[Int] = {
  val num = lst.length / 2
  if (num == 0) lst
  else {
    def merge(lst: List[Int], lst1: List[Int]): List[Int] =
        (lst, lst1) match {
        case(Nil, lst1) => lst1
        case(lst, Nil) => lst
        case(x :: xs1, y :: ys1) =>
          if (x < y) x::merge(xs1, lst1)
          else y :: merge(lst, ys1)
    }
    val (left, right) = lst splitAt(num)
    merge(mergeSort(left), mergeSort(right))
  }
}
val list1=mergeSort(List(4,63,7,34,32))
println(list1)
}
```

[ ]:
```
/**
* Implementing QuickSort using List
*/

class UsingQuickSort {
    def sort(lst:List[Int]): List[Int] =
        if (lst.length < 2) lst
        else {
            val pivot = lst(lst.length / 2)
            sort (lst filter (pivot > _)) ::: (lst filter (pivot == _)) :::␣
  ↪sort (lst filter(pivot < _))
        }
}

object QuickSort extends App {

    val quicksort = new UsingQuickSort
    val lst = List(2,1,3,5,4,9,6)
    val list = quicksort.sort(lst)
    println(list)

}
```

- implement Stack and Queue using Lists.

[ ]:
```
//queue using list

class QueueUsingList{                                  //class to implement␣
  ↪Queue using list
  def enqueue(element:Int,l: List[Int]):List[Int]={     //function to add␣
↪element to the queue
      val myList= l :+ element
      myList
  }


  def dequeue(l:List[Int])={//function to delete element from the queue
      val myList=l.drop(1)
      myList
  }
}



object Queue1 extends App{
    val obj = new QueueUsingList//Instantiating class QueueUsingList
    val queuelist1 = obj.enqueue(4,List(1,2,3))
    println(s"The list after pushing element 5 is $queuelist1")
    val queuelist2 = obj.enqueue(6,List(1,2,3,4,6))
    println(s"The list after pushing element 6 is $queuelist2")
    val dequeuelist = obj.dequeue(queuelist2)
    println(s"The list after popping last element is $dequeuelist")
}
```

[ ]:
```
/**
implement Stack using Lists.
*/
class PushPop
{
  def push(list:List[Int],value:Int)=
  {
    val list1=list
    val list2=List(value)
    val list3=list1:::list2

    list3
    }
  def pop(list:List[Int]):List[Int]=
  {
    val list1=list
    val list2=list1.init
    list2


    }

}
```

```
object Stack extends App{
  val lst=List[Int]()
  val obj1 = new PushPop()
  val newlist= obj1.push(lst, 1)
  println(newlist)
  val new1list= obj1.push(newlist, 2)
  println(new1list)
  val new2list= obj1.push(new1list, 3)
  println(new2list)

  val new3list=obj1.pop(new2list)
  println(new3list)
  val new4list=obj1.pop(new3list)
  println(new4list)
}
class PushPop
{
  def push(list:List[Int],value:Int)=
  {
    val list1=list
    val list2=List(value)
    val list3=list1:::list2

    list3
    }
  def pop(list:List[Int]):List[Int]=
  {
    val list1=list
    val list2=list1.init
    list2


    }

}
object Stack extends App{
  val lst=List[Int]()
  val obj1 = new PushPop()
  val newlist= obj1.push(lst, 1)
  println(newlist)
  val new1list= obj1.push(newlist, 2)
  println(new1list)
  val new2list= obj1.push(new1list, 3)
  println(new2list)

  val new3list=obj1.pop(new2list)
  println(new3list)
  val new4list=obj1.pop(new3list)
  println(new4list)
}
```