Scala Assignment

Problem Statement 1:

Generate solution for you are tasked with creating a random password generator in Scala. The generator will take user input for password length and generate a random password that includes a mix of lowercase letters, uppercase letters, numbers, and special characters.

```
import scala.util.Random

object PasswordGenerator{

 val lowercase = "abcdefghijklmnopqrstuvwxyz"

 val uppercase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

 val digits = "0123456789"

 val special_charas = "!@#$%^&*()_+[]{}|;:,.<>?/"


 val combined_charas = lowercase+uppercase+digits+special_charas

 def PasswordGenerator(len: Int):String ={

  if(len < 0)

  {

   println("Input value must be greater than 0")

  }

  val random = new Random()

  (1 to len).map{

   _ => combined_charas(random.nextInt(combined_charas.length))

  }.mkString

 }

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

  println("Enter the desired length of password :")

  val length = scala.io.StdIn.readLine().toInt
```

```
  if(length > 0)
  {
    val password = PasswordGenerator(length)
    println(s"Generated password is : $password")
  }
  else
  {
    println("Enter a positive value")
  }
 }
}
```

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Intel
Enter the desired length of password :
8
Generated password is : wnZ4N8&)

Process finished with exit code 0
```

<u>Problem Statement 2: UST Shopping Cart Application:</u>

You are tasked with developing a Shopping Cart application in Scala. The application will manage a shopping cart, allowing customers to add, remove, update, view items in their cart, and proceed to payment. Each item will have details such as name, quantity, price, and category. Additionally, users will be able to make payments through a simulated payment gateway – Credit Car, Debit Card, UPI. The application will also calculate the total price including GST (Goods and Services Tax) and will add delivery charges below than Rs.200 cart value.

```
import scala.collection.mutable.Map

import scala.io.StdIn


class Items(val id: Int, val name: String, var quantity: Int, val price: Double, val category:
String)
```

```
class ShoppingCart() {
 var list_item: Map[Int, Items] = Map()
 var shopItems: Map[String, Items] = Map(
  "Apple" -> new Items(1, "Apple", 0, 0.5, "Fruit"),
  "Banana" -> new Items(2, "Banana", 0, 0.3, "Fruit"),
  "Carrot" -> new Items(3, "Carrot", 0, 0.2, "Vegetable"),
  "Bread" -> new Items(4, "Bread", 0, 1.5, "Bakery"),
  "Milk" -> new Items(5, "Milk", 0, 1.0, "Dairy")
 )

 def showAvailableItems(): Unit = {
  println("Available items:")
  shopItems.values.foreach { item =>
   println(s"Name: ${item.name}, Price: ${item.price}, Category: ${item.category}")
  }
  println("Enter item details in the format: name, quantity")
 }

 def addItemFromInput(): Unit = {
  showAvailableItems()
  val input = StdIn.readLine().split(",").map(_.trim)
  try {
   if (input.length == 2) {
    val name = input(0)
    val quantity = input(1).toInt
    if (shopItems.contains(name)) {
     addItem(name, quantity)
```

```
      } else {
        println("Product not found in shop.")
      }
    } else {
      throw new IllegalArgumentException("Invalid input format.")
    }
  } catch {
    case e: NumberFormatException =>
      println("Invalid input format. Please enter numeric value for quantity.")
    case e: IllegalArgumentException =>
      println(e.getMessage)
  }
}

def addItem(name: String, quantity: Int): Unit = {
  if (quantity < 0) {
    println("Quantity cannot be negative. Item not added.")
  } else {
    val shopItem = shopItems(name)
    val newItem = new Items(shopItem.id, shopItem.name, quantity, shopItem.price,
shopItem.category)
    list_item += (shopItem.id -> newItem)
    println(s"Added ${quantity} ${shopItem.name}(s) to the cart.")
  }
}

def updateItemFromInput(): Unit = {
  println("Enter item ID to update:")
```

```
  val id = StdIn.readInt()
  if (list_item.contains(id)) {
    println(s"Enter updated item details (quantity):")
    val input = StdIn.readLine().trim
    try {
      val quantity = input.toInt
      updateItem(id, quantity)
    } catch {
      case e: NumberFormatException =>
        println("Invalid input format. Please enter numeric value for quantity.")
    }
  } else {
    println(s"Item with ID $id not found in cart.")
  }
}

def updateItem(id: Int, quantity: Int): Unit = {
  if (quantity < 0) {
    println("Quantity cannot be negative. Item not updated.")
  } else {
    val item = list_item(id)
    val updatedItem = new Items(id, item.name, quantity, item.price, item.category)
    list_item(id) = updatedItem
    println(s"Updated ${item.name} quantity to $quantity.")
  }
}

def removeItemFromInput(): Unit = {
```

```
  println("Enter item ID to remove:")
  val id = StdIn.readInt()
  removeItem(id)
 }

 def removeItem(id: Int): Unit = {
  if (list_item.contains(id)) {
   val item = list_item(id)
   list_item -= id
   println(s"Removed ${item.name} from the cart.")
  } else {
   println(s"Item with ID $id not found in cart.")
  }
 }

 def viewCart(): Unit = {
  if (list_item.isEmpty) {
   println("Cart is empty.")
  } else {
   println("Cart contents:")
   list_item.values.foreach { item =>
    println(s"ID: ${item.id}, Name: ${item.name}, Quantity: ${item.quantity}, Price: ${item.price}, Category: ${item.category}")
   }
  }
 }

 def totalPrice(withGST: Boolean = true): Unit = {
```

```
  val totalPriceWithoutGST = list_item.values.map(item => item.price *
item.quantity).sum

  val gstRate = 0.05

  val totalPriceWithGST = if (withGST) totalPriceWithoutGST * (1 + gstRate) else
totalPriceWithoutGST

  if (withGST) {

    println(f"Total Price (with GST): $totalPriceWithGST%.2f")

  } else {

    println(f"Total Price (without GST): $totalPriceWithoutGST%.2f")

  }

 }


 def processPayment(amount: Double, paymentMethod: String): Unit = {

  val paymentGateway = new PaymentGateway()

  val confirmationMessage = paymentGateway.processPayment(amount,
paymentMethod)

  println(confirmationMessage)

 }

}


class PaymentGateway {

 def processPayment(amount: Double, paymentMethod: String): String = {

  paymentMethod match {

    case "Credit Card" => s"Payment of $$${amount} processed successfully via Credit
Card."

    case "Debit Card" => s"Payment of $$${amount} processed successfully via Debit
Card."

    case "UPI" => s"Payment of $$${amount} processed successfully via UPI."

    case _ => s"Invalid payment method: $paymentMethod"
```

```
    }
  }
}

object ShoppingCartApp {
  def main(args: Array[String]): Unit = {
    val sh = new ShoppingCart()
    var continue = true
    while (continue) {
      println("\nMenu:")
      println("1. Add Item")
      println("2. Update Item")
      println("3. Remove Item")
      println("4. View Cart")
      println("5. Total Price")
      println("6. Process Payment")
      println("7. Exit")
      println("Enter your choice:")
      val choice = StdIn.readInt()
      choice match {
        case 1 => sh.addItemFromInput()
        case 2 => sh.updateItemFromInput()
        case 3 => sh.removeItemFromInput()
        case 4 => sh.viewCart()
        case 5 => sh.totalPrice()
        case 6 =>
          println("Enter payment amount:")
          val amount = StdIn.readDouble()
```

```
        println("Enter payment method (Credit Card, Debit Card, UPI):")
        val paymentMethod = StdIn.readLine().trim
        sh.processPayment(amount, paymentMethod)
      case 7 => continue = false
      case _ => println("Invalid choice. Please enter a number from 1 to 7.")
    }
  }
 }
}
```

```
Menu:
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Total Price
6. Process Payment
7. Exit
Enter your choice:
1
Available items:
Name: Apple, Price: 0.5, Category: Fruit
Name: Carrot, Price: 0.2, Category: Vegetable
Name: Milk, Price: 1.0, Category: Dairy
Name: Bread, Price: 1.5, Category: Bakery
Name: Banana, Price: 0.3, Category: Fruit
Enter item details in the format: name, quantity
```

```
Enter item details in the format: name, quantity
Bread, 4
Added 4 Bread(s) to the cart.

Menu:
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Total Price
6. Process Payment
7. Exit
Enter your choice:
4
Cart contents:
ID: 4, Name: Bread, Quantity: 4, Price: 1.5, Category: Bakery
```

```
Menu:
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Total Price
6. Process Payment
7. Exit
Enter your choice:
2
Enter item ID to update:
4
Enter updated item details (quantity):
5
Updated Bread quantity to 5.
```

```
Enter your choice:
4
Cart contents:
ID: 4, Name: Bread, Quantity: 5, Price: 1.5, Category: Bakery

Menu:
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Total Price
6. Process Payment
7. Exit
Enter your choice:
5
Total Price (with GST): 7.88
```

```
Menu:
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Total Price
6. Process Payment
7. Exit
Enter your choice:
6
Enter payment amount:
7.88
Enter payment method (Credit Card, Debit Card, UPI):
UPI
Payment of $7.88 processed successfully via UPI.
```

Problem Statement 3: Case Classes and Pattern Matching

Create a Scala application that uses case classes to model a simple payroll system. Implement pattern matching to calculate the salary of different types of employee – FullTimeEmployee, PartTimeEmployee, ContractType, Freelancers.

```
object PayrollSystem {
 class Employee(val name: String)
 case class FullTimeEmployee(override val name: String, salary: Double) extends
Employee(name)
 case class PartTimeEmployee(override val name: String, hoursWorked: Int, rate:
Double) extends Employee(name)
 case class ContractEmployee(override val name: String, hoursWorked: Int, rate:
Double) extends Employee(name)
 case class Freelancer(override val name: String, hoursWorked: Int, rate: Double)
extends Employee(name)
 def calculateSalary(employee: Employee): Double = employee match {
 case FullTimeEmployee(_, salary) => salary
 case PartTimeEmployee(_, hoursWorked, rate) => hoursWorked * rate
 case ContractEmployee(_, hoursWorked, rate) => hoursWorked * rate
 case Freelancer(_, hoursWorked, rate) => hoursWorked * rate
 }
 def main(args: Array[String]): Unit = {
 val employees = List(
 FullTimeEmployee("Tom", 50000),
 PartTimeEmployee("Peter Griffin", 20, 15.0),
 ContractEmployee("Liana", 40, 25.0),
 Freelancer("Lois Griffin", 50, 30.0)
 )
 employees.foreach { employee =>
```

```
 val salary = calculateSalary(employee)

 println(s"The salary of ${employee.name} is $salary")

 }

 }

}
```

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagen
Tom has a salary of 50000.0.
Peter Griffin has a salary of 300.0.
Liana has a salary of 1000.0.
Lois Griifin has a salary of 1500.0.

Process finished with exit code 0
```

Problem Statement 4: File Processing

Write a Scala program to read a text file, count the occurrences of each word, and display the top N most frequent words.

Create a method wordCount(filePath: String, topN: Int): List[(String, Int)] that reads a text file and returns a list of tuples containing the top N most frequent words and their counts.

Program ask user to enter N top most frequent words and show N most frequent words as output.

```
import scala.io.StdIn

import scala.collection.MapView

import scala.io.Source


object wordCount {

  def wordCount(filePath: String): MapView[String, Int] = {

    val file =
Source.fromFile("C:\\Users\\Administrator\\Documents\\Training\\tourism.txt")

    val lines = file.getLines().toList

    file.close()
```

```
  val wordCounts = lines
    .flatMap(_.split("\\s+"))
    .filter(_.nonEmpty)
    .groupBy(_.toLowerCase)
    .mapValues(_.size)
  wordCounts
 }
 def N_freq_words(wordCounts: MapView[String, Int], n: Int): List[(String, Int)] = {
  wordCounts.toList.sortBy(-_._2).take(n)
 }
 def main(args: Array[String]): Unit = {
  val filePath = "C:\\Users\\Administrator\\Documents\\Training\\tourism.txt"
  println("Enter the number of top frequent words you wish to see : ")
  val n = StdIn.readInt()
  val count = wordCount(filePath)
  val topwords = N_freq_words(count, n)
  println(s"\nTop $n frequent words in the file '$filePath' are :")
  topwords.foreach { case (word, count) =>
    println(s"$word: $count")
  }
 }
}
```

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
Enter the number of top frequent words you wish to see :
7

Top 7 frequent words in the file 'C:\Users\Administrator\Documents\Training\tourism.txt' are :
and: 43
the: 24
is: 18
for: 16
a: 16
of: 15
in: 13

Process finished with exit code 0
```

Problem Statement 5: File Analysis Application in Scala

The application will process a text file and provide various analytical insights about its content. The insights will include word count, line count, character count, frequency of each word, and the top N most frequent words.

FileAnalyzer Class: Create a FileAnalyzer class with the following methods:

loadFile(filePath: String): Load and Read a text file.

wordCount(): Returns the total number of words in the file.

lineCount(): Returns the total number of lines in the file.

characterCount(): Returns the total number of characters in the file.

averageWordLength(): Double: Returns the average word length in the file.

mostCommonStartingLetter(): Option[Char]: Returns the most common starting alphabet of words in the input files.

wordOccurrences(word: String): Int: Returns the number of occurrences of a specific word in file.

```
import scala.io.Source

import scala.collection.MapView
```

```
class FileAnalyzer(filePath: String) {
  private val source =
Source.fromFile("C:\\Users\\Administrator\\Documents\\Training\\tourism.txt")
  private val lines = source.getLines().toList
  private val charas = lines.mkString
  private val words = lines.flatMap(_.split("\\s+"))
  def close(): Unit = source.close()
  def loadFile(): Unit = {}
  def wordCount(): Int = words.length
  def lineCount(): Int = lines.length
  def characterCount(): Int = charas.length
  def averageWordLength(): Double = {
    if (words.isEmpty) 0.0
    else words.map(_.length).sum.toDouble / words.length
  }
  def mostCommonStartingLetter(): Option[Char] = {
    if (words.isEmpty) None
    else {
      val startingLetters = words.map(_.headOption.map(_.toLower)).collect { case
        Some(c) => c }
      if (startingLetters.isEmpty) None
      else {
        val grouped = startingLetters.groupBy(identity).view.mapValues(_.size)
        grouped.maxByOption(_._2).map(_._1)
      }
    }
  }
```

```
  def wordOccurrences(word: String): Int = {
    words.count(_.equalsIgnoreCase(word))
  }
}
def N_freq_words(wordCounts: MapView[String, Int], n: Int): List[(String, Int)] = {
  wordCounts.toList.sortBy(-_._2).take(n)
}
object Analyzer {
  def main(args: Array[String]): Unit = {
    val analyzer = new
        FileAnalyzer("C:\\Users\\Administrator\\Downloads\\dictionary_data.txt")
    analyzer.loadFile()
    println(s"Word Count: ${analyzer.wordCount()}")
    println(s"Line Count: ${analyzer.lineCount()}")
    println(s"Character Count: ${analyzer.characterCount()}")
    println(s"Average Word Length: ${analyzer.averageWordLength()}")
    println(s"Most Common Starting Letter: ${analyzer.mostCommonStartingLetter()}")
    println("Enter the word to find its number of occurances : ")
    val word = scala.io.StdIn.readLine()
    println(s"Occurrences of $word : ${analyzer.wordOccurrences(word)}")
    analyzer.close()
  }
}
```

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetB
Word Count: 664
Line Count: 11
Character Count: 4617
Average Word Length: 5.969879518072289
Most Common Starting Letter: Some(a)
Enter the word to find its number of occurances :
kerala
Occurrences of kerala : 9

Process finished with exit code 0
```