SCALA ASSIGNMENT

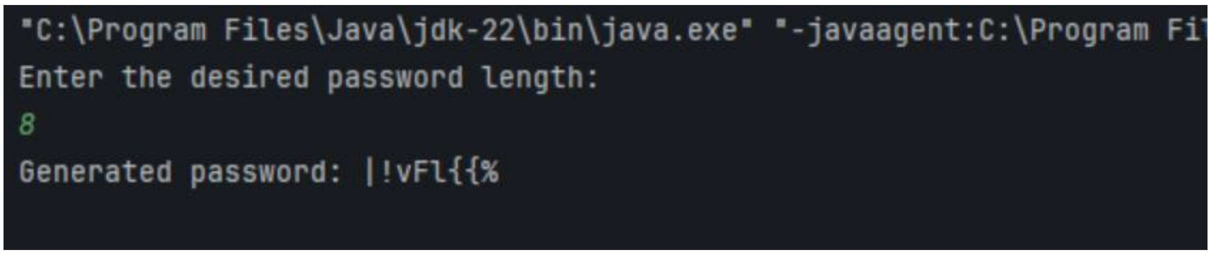
Problem Statement 1:

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.

CODE

```
import scala.util.Random
object PassworGenerator{
 val lower = "abcdefghijklmnopqrstuvwxyz"
 val upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 val digits = "0123456789"
 val specialChars = "!@#$%^&*()_+[]{}|;:,.<>?/"


 val allchars = lower+upper+digits+specialChars
 def GeneratePassword(length: Int):String ={
  if(length < 0)
  {
   println("Password length must be greater than 0")
  }
  val random = new Random()
  (1 to length).map{
   _ => allchars(random.nextInt(allchars.length))
  }.mkString
 }
 def main(args: Array[String]): Unit = {
  println("Enter the desired password length:")
  val input = scala.io.StdIn.readLine()
  val length = input.toInt
  if(length > 0)
  {
   val password = GeneratePassword(length)
   println(s"Generated password: $password")
  }
  else
  {
   println("Enter a positive integer")
  }
 }
}
```

<u>OUTPUT</u>



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Fi
Enter the desired password length:
8
Generated password: |!vFl{{%
```

Problem Statement 2: UST Shopping Cart Application

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.

<u>Item Class:</u>

Create an Item () class with the following attributes:

id: Unique identifier for the item

name: Name of the item.

quantity: Quantity of the item.

price: Price of the item.

category: Category of the item.

Shopping Cart:

Create a ShoppingCart() class that manages a collection of Item objects. Implement the following methods:

addItem(item: Item): Adds a new item to the cart.

updateItem(id: Int, updatedItem: Item): Updates an existing item in the cart.

removeItem(id: Int): Removes an item from the cart.

viewCart(): Displays all items in the cart.

totalPrice(withGST: Boolean = true): Calculates and displays the total price of all items in the cart with GST charges.

Payment Gateway:

Create a PaymentGateway class to simulate payment processing. Implement the following methods:

processPayment(amount: Double, paymentMethod: String): Simulates processing a payment and returns a confirmation message.

Payment methods can include "Credit Card", "Debit Card", and "UPI".

GST Calculation:

Assume a GST rate of 5%. Implement the GST calculation within the totalPrice method.

Cart Operations:

Implement the following functions:

Add Item: Allow users to input details for a new item and add it to the cart.

Update Item: Allow users to update the details of an existing item.

Remove Item: Allow users to remove an item from the cart.

View Cart: Display all items in the cart.

Calculate Total: Calculate and display the total price of items in the cart, optionally including GST.

Make Payment: Allow users to proceed to payment and select a payment method. Use the PaymentGateway class to process the payment.

Error Handling:

Ensure appropriate error handling for scenarios such as trying to update or remove an item that doesn't exist, and handling payment errors.

Sample Outcome:

```
Welcome to the UST Shopping Cart!


 Please choose an option:
 1. Add a new item
 2. Update an existing item
 3. Remove an item
 4. View cart
 5. Calculate total price
 6. Make payment
 7. Exit


 Option: 1
 Enter item name: Apple
 Enter quantity: 3
 Enter price: 50.00
 Enter category: Fruits & Vegetables
 Item added successfully!


 Option: 4
 Viewing cart:
 1. [1] Apple - Quantity: 3, Price: 50.00, Category: Fruits & Vegetables
 ...
 Option: 5
 Cart Value:                ₹150.00
 Deliver Charge:                   ₹30.00
 GST:                         ₹7.50
 Amount Payable:                  ₹187.50
```

```
 Option: 6
 Choose a payment method (Credit Card/Debit Card/UPI): Credit Card
 Processing payment of 187.50 using Credit Card...
 Payment successful! Confirmation number: UST123456789
```

```
 Option: 7
 Exiting the application. Goodbye!
```

CODE

```
import scala.collection.mutable

case class Item(id: Int, name: String, var quantity: Int, price: Double, category: String)

class ShoppingCart {
 private val items = mutable.Map[String, Item]()
 private val gstRate = 0.05
 private val deliveryChargeThreshold = 200
 private val deliveryCharge = 2
```

```
def addItem(item: Item): Unit = {
  if (items.contains(item.name.toLowerCase)) {
    val existingItem = items(item.name.toLowerCase)
    existingItem.quantity += item.quantity
    if (existingItem.quantity == 0) {
      removeItem(item.name)
    } else {
      println(s"Item ${item.name} quantity updated in the cart.")
    }
  } else {
    items(item.name.toLowerCase) = item
    println(s"Item ${item.name} added to the cart.")
  }
}


def updateItem(name: String, newQuantity: Int): Unit = {
  if (items.contains(name.toLowerCase)) {
    val existingItem = items(name.toLowerCase)
    existingItem.quantity = newQuantity
    if (existingItem.quantity == 0) {
      removeItem(name)
    } else {
      println(s"Item ${name} updated. Quantity is now ${newQuantity}.")
    }
  } else {
    println(s"Item ${name} does not exist.")
  }
}
```

```
def removeItem(name: String): Unit = {
  if (items.contains(name.toLowerCase)) {
    items.remove(name.toLowerCase)
    println(s"Item ${name} removed from the cart.")
  } else {
    println(s"Item ${name} does not exist.")
  }
}


def viewCart(): Unit = {
  if (items.isEmpty) {
    println("Your cart is empty.")
  } else {
    println("Items in your cart:")
    items.values.foreach { item =>
      val totalPrice = item.quantity * item.price
      println(s"${item.name} X${item.quantity} - ${totalPrice}")
    }
  }
}


def totalPrice(withGST: Boolean = true): Option[Double] = {
  if (items.isEmpty) {
    println("Your cart is empty. Add items to calculate the total.")
    None
  } else {
    val subtotal = items.values.map(item => item.quantity * item.price).sum
    val gst = if (withGST) subtotal * gstRate else 0
    val total = subtotal + gst
    val finalTotal = if (total < deliveryChargeThreshold) total + deliveryCharge else total
    println(f"Subtotal: $$${subtotal}%.2f, GST: $$${gst}%.2f, Total: $$${finalTotal}%.2f")
    Some(finalTotal)
  }
 }
}


class PaymentGateway {
 def processPayment(amount: Double, paymentMethod: String): String = {
  paymentMethod match {
    case "Credit Card" | "Debit Card" | "UPI" =>
     s"Payment of $$${amount} processed successfully using $paymentMethod."
    case _ =>
     s"Payment method $paymentMethod not supported."
  }
 }
}


// Main Application
object ShoppingCartApp extends App {
 val presetItems = List(
  Item(1, "Apple", 0, 0.5, "Fruit"),
  Item(2, "Banana", 0, 0.3, "Fruit"),
  Item(3, "Bread", 0, 2.0, "Bakery"),
  Item(4, "Milk", 0, 1.5, "Dairy")
 )
```

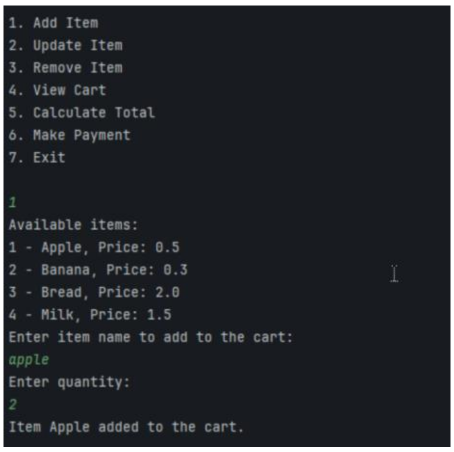
```
val cart = new ShoppingCart
val paymentGateway = new PaymentGateway


def menu(): Unit = {
  println(
    """
      |1. Add Item
      |2. Update Item
      |3. Remove Item
      |4. View Cart
      |5. Calculate Total
      |6. Make Payment
      |7. Exit
      |""".stripMargin)
}


def displayPresetItems(): Unit = {
  println("Available items:")
  presetItems.foreach { item =>
    println(s"${item.id} - ${item.name}, Price: ${item.price}")
  }
}


var exit = false


while (!exit) {
```

```
menu()

val choice = scala.io.StdIn.readInt()



choice match {

 case 1 =>

  displayPresetItems()

  println("Enter item name to add to the cart:")

  val name = scala.io.StdIn.readLine().trim

  presetItems.find(_.name.equalsIgnoreCase(name)) match {

   case Some(item) =>

    println("Enter quantity:")

    val quantity = scala.io.StdIn.readInt()

    if (quantity < 1) {

     println("Please enter a positive quantity")

    }

    else

     {

      cart.addItem(item.copy(quantity = quantity))

     }

   case None =>

    println("Invalid item name.")

  }



 case 2 =>

  println("Enter item name to update:")

  val name = scala.io.StdIn.readLine().trim

  presetItems.find(_.name.equalsIgnoreCase(name)) match {

   case Some(_) =>

    println("Enter new quantity:")
```

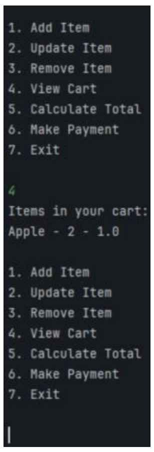
```
      val quantity = scala.io.StdIn.readInt()
      if(quantity == 0)
       {
         cart.removeItem(name)
       }
      else if(quantity < 0){
       println("Enter a positive quantity")
      }
      else
       {
         cart.updateItem(name, quantity)
       }
    case None =>
      println("Invalid item name.")
  }


case 3 =>
  println("Enter item name to remove:")
  val name = scala.io.StdIn.readLine().trim
  cart.removeItem(name)


case 4 =>
  cart.viewCart()


case 5 =>
  cart.totalPrice() match {
    case Some(total) =>
      println("Select payment method (Credit Card, Debit Card, UPI):")
```

```
          val paymentMethod = scala.io.StdIn.readLine()

          val confirmation = paymentGateway.processPayment(total, paymentMethod)

          println(confirmation)

        case None =>

      }



    case 6 =>

      println("Total needs to be calculated before making payment.")



    case 7 =>

      exit = true



    case _ =>

      println("Invalid option. Please try again.")

  }

 }

}
```

OUTPUT



```
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Calculate Total
6. Make Payment
7. Exit

1
Available items:
1 - Apple, Price: 0.5
2 - Banana, Price: 0.3
3 - Bread, Price: 2.0
4 - Milk, Price: 1.5
Enter item name to add to the cart:
apple
Enter quantity:
2
Item Apple added to the cart.
```



```
1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Calculate Total
6. Make Payment
7. Exit

4
Items in your cart:
Apple - 2 - 1.0

1. Add Item
2. Update Item
3. Remove Item
4. View Cart
5. Calculate Total
6. Make Payment
7. Exit
```

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.

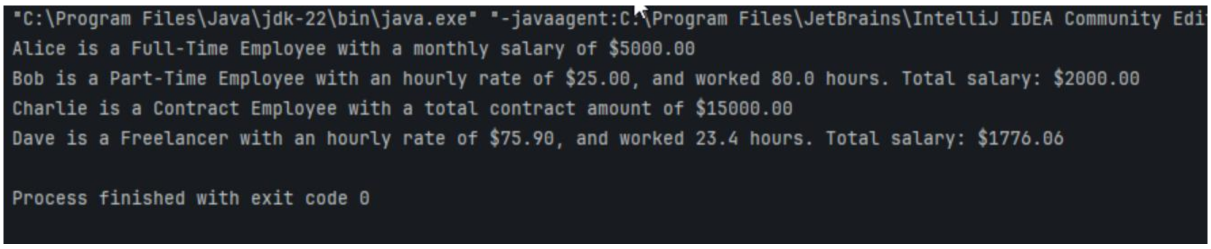
CODE

```
object PayrollSystem {

 sealed trait Employee {
  def calculateSalary: Double
 }



 case class FullTimeEmployee(name: String, monthlySalary: Double) extends Employee {
  def calculateSalary: Double = monthlySalary
 }

 case class PartTimeEmployee(name: String, hourlyRate: Double, hoursWorked: Double)
extends Employee {
  def calculateSalary: Double = hourlyRate * hoursWorked
 }

 case class ContractEmployee(name: String, contractAmount: Double) extends Employee
{
  def calculateSalary: Double = contractAmount
 }

 case class Freelancer(name: String, hourlyRate: Double, hoursWorked: Double) extends
Employee {
  def calculateSalary: Double = hourlyRate * hoursWorked
 }

 def displaySalary(employee: Employee): Unit = {
  employee match {
   case FullTimeEmployee(name, monthlySalary) =>
    println(s"$name is a Full-Time Employee with a monthly salary of
$$${monthlySalary.formatted("%.2f")}")
   case PartTimeEmployee(name, hourlyRate, hoursWorked) =>
    val totalSalary = hourlyRate * hoursWorked
    println(s"$name is a Part-Time Employee with an hourly rate of
$$${hourlyRate.formatted("%.2f")}, " +
     s"and worked $hoursWorked hours. Total salary: $$${totalSalary.formatted("%.2f")}")
   case ContractEmployee(name, contractAmount) =>
    println(s"$name is a Contract Employee with a total contract amount of
$$${contractAmount.formatted("%.2f")}")
   case Freelancer(name, hourlyRate, hoursWorked) =>
    val totalSalary = hourlyRate * hoursWorked
    println(s"$name is a Freelancer with an hourly rate of
$$${hourlyRate.formatted("%.2f")}, " +
     s"and worked $hoursWorked hours. Total salary: $$${totalSalary.formatted("%.2f")}")
  }
 }

 def main(args: Array[String]): Unit = {
  val employees: List[Employee] = List(
   FullTimeEmployee("Alice", 5000.0),
   PartTimeEmployee("Bob", 25.0, 80.0),
   ContractEmployee("Charlie", 15000.0),
   Freelancer("Dave", 75.9, 23.4)
  )

  employees.foreach(displaySalary)
```

 }

}

OUTPUT



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edi
Alice is a Full-Time Employee with a monthly salary of $5000.00
Bob is a Part-Time Employee with an hourly rate of $25.00, and worked 80.0 hours. Total salary: $2000.00
Charlie is a Contract Employee with a total contract amount of $15000.00
Dave is a Freelancer with an hourly rate of $75.90, and worked 23.4 hours. Total salary: $1776.06

Process finished with exit code 0
```

<u>Problem Statement 4: File Processing</u>

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.

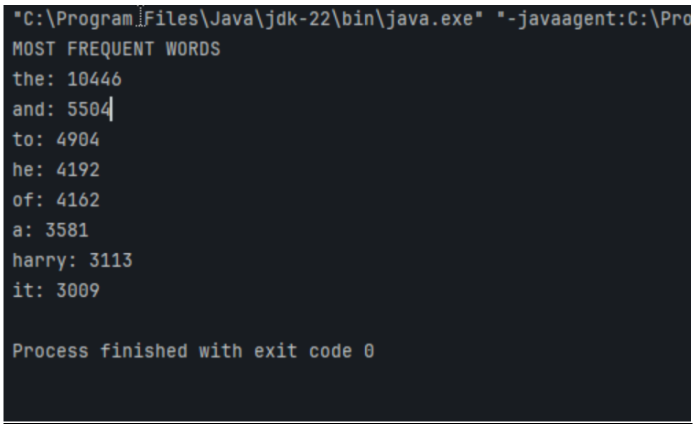
Program ask user to enter N top most frequent words and show N most frequent words as output.

<u>CODE</u>

```
import scala.io.Source

import scala.collection.mutable


object wordFrequencyAnalyzer {

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

  if (topN <= 0) {

   println("The number of top words must be greater than 0")

   return List()

  }


  val file = new java.io.File(filePath)

  if (!file.exists) {

   println(s"File not found: $filePath")
```

```
      return List()
    }

    val wordcount = mutable.Map[String, Int]().withDefaultValue(0)
    for (line <- Source.fromFile(filePath).getLines()) {
      val words = line.toLowerCase.split("\\W+").filter(_.nonEmpty)
      for (word <- words) {
        wordcount(word) += 1
      }
    }

    wordcount.toList.sortBy(-_._2).take(topN)
  }

  def main(args: Array[String]): Unit = {
    val filepath = "C:\\Users\\Administrator\\Downloads\\Harry Potter and the Deathly
Hallows.txt"
    val topWords = wordCount(filepath, 8)
    if (topWords.nonEmpty) {
      println("MOST FREQUENT WORDS")
      topWords.foreach { case (word, count) => println(s"$word: $count") }
    }
  }
}
```

OUTPUT


```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Pro
MOST FREQUENT WORDS
the: 10446
and: 5504
to: 4904
he: 4192
of: 4162
a: 3581
harry: 3113
it: 3009

Process finished with exit code 0
```

<u>Problem Statement 5: File Analysis Application in Scala</u>

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.

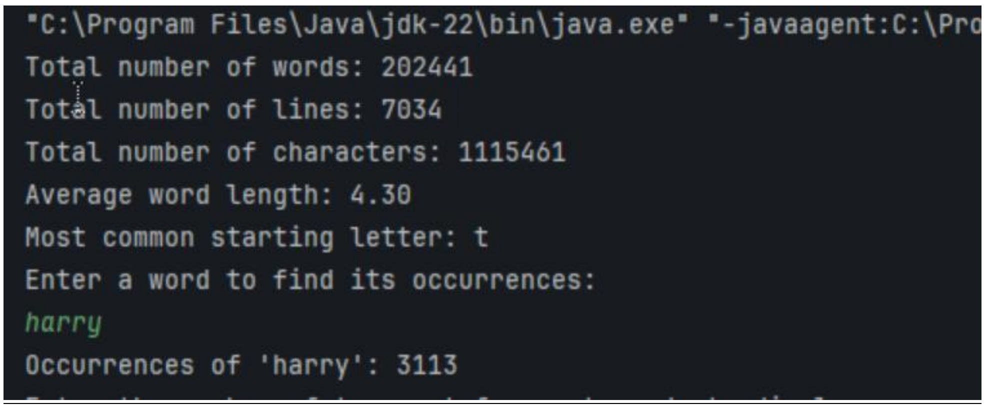
wordOccurrences(word: String): Int: Returns the number of occurrences of a specific word in file.

CODE

```
import scala.io.Source

import scala.collection.mutable



class FileAnalyzer(filePath: String) {

 private val lines = Source.fromFile(filePath).getLines().toList
 private val words = lines.flatMap(_.toLowerCase.split("\\W+").filter(_.nonEmpty))
 private val wordCounts = mutable.Map[String, Int]().withDefaultValue(0)
 words.foreach(word => wordCounts(word) += 1)


 def wordCount(): Int = words.size


 def lineCount(): Int = lines.size


 def characterCount(): Int = lines.mkString("").length


 def averageWordLength(): Double = if (words.isEmpty) 0.0 else
words.map(_.length).sum.toDouble / words.size


 def mostCommonStartingLetter(): Option[Char] = {
   val startingLetters = words.map(_.headOption).flatten
   if (startingLetters.isEmpty) None
   else Some(startingLetters.groupBy(identity).maxBy(_._2.size)._1)
 }


 def wordOccurrences(word: String): Int = wordCounts(word.toLowerCase)


 def topNMostFrequentWords(topN: Int): List[(String, Int)] = {
   wordCounts.toList.sortBy(-_._2).take(topN)
 }
```

```
}

object FileAnalyzerApp {
 def main(args: Array[String]): Unit = {
  val filePath = "C:\\Users\\Administrator\\Downloads\\Harry Potter and the Deathly Hallows.txt"

  val analyzer = new FileAnalyzer(filePath)

  println(s"Total number of words: ${analyzer.wordCount()}")
  println(s"Total number of lines: ${analyzer.lineCount()}")
  println(s"Total number of characters: ${analyzer.characterCount()}")
  println(f"Average word length: ${analyzer.averageWordLength()}%.2f")
  println(s"Most common starting letter:
${analyzer.mostCommonStartingLetter().getOrElse("None")}")

  println("Enter a word to find its occurrences:")
  val word = scala.io.StdIn.readLine()
  println(s"Occurrences of '$word': ${analyzer.wordOccurrences(word)}")

  println("Enter the number of top most frequent words to display:")
  val topN = scala.io.StdIn.readInt()
  println(s"Top $topN most frequent words:")
  analyzer.topNMostFrequentWords(topN).foreach { case (word, count) =>
   println(s"$word: $count")
  }
 }}
```

OUTPUT



```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Pro
Total number of words: 202441
Total number of lines: 7034
Total number of characters: 1115461
Average word length: 4.30
Most common starting letter: t
Enter a word to find its occurrences:
harry
Occurrences of 'harry': 3113
```