Kingdom of Saudi Arabia
Ministry of Education
University of Jeddah
College of Science and Computer Engineering

المملكة العربية السعودية
وزارة التعليم
جامعة جَدَة
كلية علوم و هندسة الحاس ب

Writing Spark Jobs!

Amal Almutairi

Nouf Mansour

Noor Balfas

Muna Saleh Dini

Kingdom of Saudi Arabia
Ministry of Education
University of Jeddah
College of Science and Computer Engineering

المملكة العربية السعودية
وزارة التعليم
جامعة جَدَة
كلية علوم و هندسة الحاس ب

First question :

Step 1: Creating the data set

```
scala> val employees = sc.parallelize(Seq(
     |     ("Eman", "Abdulaziz", 1991, "F", 9000),
     |     ("Hamed", "Ryan", 2000, "M", 1200),
     |     ("Fatima", "Saeed", 1978, "F", 13000),
     |     ("Rahaf", "Abdullah", 1967, "F", 14000),
     |     ("Ahmad", "Mohamed", 1988, "M", 15000)
     | ))
employees: org.apache.spark.rdd.RDD[(String, String, Int, String, Int)] = ParallelCollectionRDD[0] at parallelize at <console>:24
```

Step 2: Select employees whose age is greater than 33

```
scala> val employeesAbove33 = employees.filter { case (_, _, birthYear, _, _) => 2024 - birthYear > 33 }
employeesAbove33: org.apache.spark.rdd.RDD[(String, String, Int, String, Int)] = MapPartitionsRDD[1] at filter at <console>:25

scala> employeesAbove33.collect().foreach(println)
[Stage 0:>

(Fatima,Saeed,1978,F,13000)
(Rahaf,Abdullah,1967,F,14000)
(Ahmad,Mohamed,1988,M,15000)
```

Step 3: Report the age of the oldest female

```
scala> val ages = employees.map { case (_, _, birthYear, _, _) => currentYear
 - birthYear }
ages: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[1] at map at <console>
:27

scala> val averageAge = ages.mean()
[Stage 0:>

averageAge: Double = 40.8

scala> println(s"Average Age: $averageAge")
Average Age: 40.8
```

Step 4: Report the number of female and male employees

```
scala> val genderCount = employees.map { case (_, _, _, gender, _) => (gender, 1) }.reduceByKey(_ + _)
genderCount: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[5] at reduceByKey at <console>:25

scala> genderCount.collect().foreach { case (gender, count) => println(s"$gender: $count") }
F: 3
M: 2
```

Step 5: Find the average Salary

```
scala> val salaries = employees.map { case (_, _, _, _, salary) => salary }
salaries: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[7] at map at <cons
ole>:25

scala> val averageSalary = salaries.mean()
averageSalary: Double = 10440.0

scala> println(s"Average Salary: $averageSalary")
Average Salary: 10440.0
```

Step 6 : Report the employees with salary less than10,000 as <Salary,LastName>

```
scala> val lowSalaryEmployees = employees
lowSalaryEmployees: org.apache.spark.rdd.RDD[(String, String, Int, String, Int)] = ParallelCollectionRDD[8] at parallelize at <consol
e>:24

scala>   .filter { case (_, _, _, _, salary) => salary < 10000 }
res7: org.apache.spark.rdd.RDD[(String, String, Int, String, Int)] = MapPartitionsRDD[8] at filter at <console>:26

scala>   .map { case (_, lastName, _, _, salary) => (salary, lastName) }
res8: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[9] at map at <console>:26

scala> lowSalaryEmployees.collect().foreach(println)
(Eman,Abdulaziz,1991,F,9000)
(Hamed,Ryan,2000,M,1200)
(Fatima,Saeed,1978,F,13000)
(Rahaf,Abdullah,1967,F,14000)
(Ahmad,Mohamed,1980,M,15000)

scala> res8.collect().foreach(println)
(9000,Abdulaziz)
(1200,Ryan)
```

Second question :

Step 1: Creating the data set

```
scala> val numbers = sc.parallelize(List(45, 3, 4, 44, 39, 11, 7, 8, 13, 21, 20, 44, 44, 12, 27, 27, 29, 18, 19, 19, 1, 1, 31, 31, 32, 1, 22, 33, 31, 37, 50
, 41, 42))
numbers: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:24
```

Step 2: Categorize Each Number into a Range

< [1, 10], [11, 20], [21, 30], [31, 40], [41, 50]>.

```
scala> val categorizeByRange = (num: Int) => {
     |   if (num >= 1 && num <= 10) "1-10"
     |   else if (num >= 11 && num <= 20) "11-20"
     |   else if (num >= 21 && num <= 30) "21-30"
     |   else if (num >= 31 && num <= 40) "31-40"
     |   else "41-50"
     | }
categorizeByRange: Int => String = $Lambda$2136/584393428@6409a2e4
```

Step 3: Map Values to Their Ranges with Initial Count

```
scala> val mapped = numbers.map(num => (categorizeByRange(num), (num, 1)))
mapped: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[1] at map at <console>:27
```

Step 4: Aggregate Sum and Count for Each Range

```
scala> val aggregated = mapped.reduceByKey { case ((sum1, count1), (sum2, count2)) => (sum1 + sum2, count1 + count2) }
aggregated: org.apache.spark.rdd.RDD[(String, (Int, Int))] = ShuffledRDD[2] at reduceByKey at <console>:25
```

Step 5: Calculate Average for Each Range

```
scala> val result = aggregated.mapValues { case (sum, count) => (count, sum.toDouble / count) }
result: org.apache.spark.rdd.RDD[(String, (Int, Double))] = MapPartitionsRDD[3] at mapValues at <console>:25
```

Step 6: Sort the Result and Collect:

```
scala> val output = result.collect().sortBy(_._1)
output: Array[(String, (Int, Double))] = Array((1-10,(7,3.5714285714285716)), (11-20,(7,16.0)), (21-30,(5,25.2)), (31-40,(7,33.42857142857143)), (41-50,(7,44.285714285714285)))
```

The Output :

```
scala> output.foreach { case (range, (count, avg)) =>
     |   println(s"Range: $range, Count: $count, Average: $avg")
     | }
Range: 1-10, Count: 7, Average: 3.5714285714285716
Range: 11-20, Count: 7, Average: 16.0
Range: 21-30, Count: 5, Average: 25.2
Range: 31-40, Count: 7, Average: 33.42857142857143
Range: 41-50, Count: 7, Average: 44.285714285714285

scala>
```

Third Question :

Step 1: Create an RDD from a list of words

```
scala> val words = sc.parallelize(List("Apple", "Orange", "Oracle", "Umbrella", "Unit", "Illness", "Early", "Artistic", "Iconic", "Idol", "Book", "Novel"))
words: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at <console>:24
```

Step 2: Filter and group words by their starting vowel

```
scala> val vowels = Set('a', 'e', 'i', 'o', 'u')
vowels: scala.collection.immutable.Set[Char] = Set(e, u, a, i, o)

scala> val groupedWords = words.filter(word => vowels.contains(word.toLowerCase.charAt(0))).groupBy(word => word.toLower
Case.charAt(0))
groupedWords: org.apache.spark.rdd.RDD[(Char, Iterable[String])] = ShuffledRDD[3] at groupBy at <console>:27
```

Step 3: Calculate total length and count for each group, then compute the average

```
scala> val wordCounts = groupedWords.mapValues(words => words.size)
wordCounts: org.apache.spark.rdd.RDD[(Char, Int)] = MapPartitionsRDD[4] at mapValues at <console>:27
→ ~
scala> val totalLengths = groupedWords.mapValues(words => words.map(_.length).sum)
totalLengths: org.apache.spark.rdd.RDD[(Char, Int)] = MapPartitionsRDD[5] at mapValues at <console>:27

scala> val averageLengths = totalLengths.join(wordCounts).mapValues { case (totalLength, count) => totalLength.toDouble
/ count }
averageLengths: org.apache.spark.rdd.RDD[(Char, Double)] = MapPartitionsRDD[9] at mapValues at <console>:27
```

Step 4: Collect and print the results

```
scala> val results = averageLengths.collect(); results.foreach { case (vowel, avgLength) => println(s"Average length of
words starting with '$vowel' : $avgLength") }
Average length of words starting with 'e' : 5.0
Average length of words starting with 'a' : 6.5
Average length of words starting with 'i' : 5.666666666666667
Average length of words starting with 'u' : 6.0
Average length of words starting with 'o' : 6.0
results: Array[(Char, Double)] = Array((e,5.0), (a,6.5), (i,5.666666666666667), (u,6.0), (o,6.0))
```