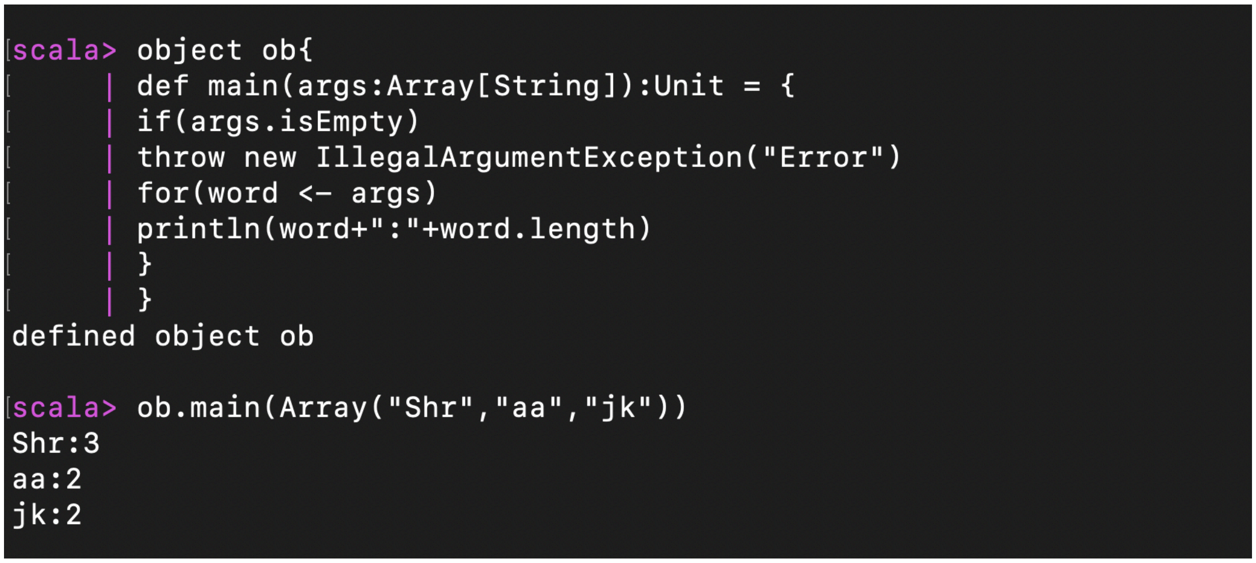
Q1.Bubble Sort

```
object ob1{
     def main(args: Array[Int]): Array[Int]= {
     var n = args.length;
     for(i<-0 until n){
     for(j<-0 until n - 1 -i){
     if(args(j) > args(j+1)){
     var temp = args(j);
     args(j)  = args(j+1);
     args(j+1) = temp;
    }
    }
    }
    args;
    }
    }
```

Output call

```
ob1.main(Array(3,1,2,5))
```

Q2.



```
[scala> object ob{
[     | def main(args:Array[String]):Unit = {
[     | if(args.isEmpty)
[     | throw new IllegalArgumentException("Error")
[     | for(word <- args)
[     | println(word+":"+word.length)
[     | }
[     | }
defined object ob

[scala> ob.main(Array("Shr","aa","jk"))
Shr:3
aa:2
jk:2
```

Q3.

val books = Seq(
  ("Dr. Seuss", "How the Grinch Stole Christmas!"),
  ("Jon Stone", "Monsters at the End of This Book"),
  ("Dr. Seuss", "The Lorax"),
  ("Jon Stone", "Big Bird in China"),
  ("Dr. Seuss", "One Fish, Two Fish, Red Fish, Blue Fish")
)
val authorCounts = books.groupBy(_._1).mapValues(_.size)

Explanation:

Explanation

1. `books.groupBy(_._1)`

books: This is the original sequence of tuples, where each tuple consists of an author's name and a book title.

scala

Copy code

```
val books = Seq(
  ("Dr. Seuss", "How the Grinch Stole Christmas!"),
  ("Jon Stone", "Monsters at the End of This Book"),
  ("Dr. Seuss", "The Lorax"),
  ("Jon Stone", "Big Bird in China"),
  ("Dr. Seuss", "One Fish, Two Fish, Red Fish, Blue Fish")
)
```

-

- **groupBy(_._1)**: The `groupBy` method groups elements of the sequence into a map. The key for the map is determined by the function passed to `groupBy`. In this case, the function is `(_._1)`.
  - `_._1`: This is a shorthand for accessing the first element of each tuple (i.e., the author's name).

After `groupBy(_._1)`, the sequence `books` is transformed into a map where each key is an author's name and each value is a sequence of tuples (books) written by that author.
scala
Copy code

```
val groupedByAuthor = books.groupBy(_._1)
// Result:
// Map(
//   "Dr. Seuss" -> Seq(("Dr. Seuss", "How the Grinch Stole
Christmas!"), ("Dr. Seuss", "The Lorax"), ("Dr. Seuss", "One Fish, Two
Fish, Red Fish, Blue Fish")),
//   "Jon Stone" -> Seq(("Jon Stone", "Monsters at the End of This
Book"), ("Jon Stone", "Big Bird in China"))
// )
```

-

2. `mapValues(_.size)`

- **mapValues(_.size)**: The `mapValues` method applies a function to each value in the map without changing the keys.
  - `_.size`: This function takes a sequence (the value in the map) and returns its size (i.e., the number of books written by the author).

After applying `mapValues(_.size)`, each sequence of tuples (books) is replaced by the count of books (i.e., the size of the sequence).
scala
Copy code

```
val authorCounts = groupedByAuthor.mapValues(_.size)
// Result:
// Map(
//   "Dr. Seuss" -> 3,
//   "Jon Stone" -> 2

      // )
```

Question 4

```
// Step 1: Define the abstract class Notification

abstract class Notification

// Step 2: Define the case classes Email and SMS that extend
Notification

case class Email(sender: String, title: String, body: String) extends
Notification

case class SMS(caller: String, message: String) extends Notification


// Step 3: Define the function showNotification with pattern matching

def showNotification(notification: Notification): String = {

  notification match {

    case Email(sender, title, _) =>

      s"You got an email from $sender with title: $title"

    case SMS(caller, message) =>

      s"You got an SMS from $caller! Message: $message"

  }

}

// Step 4: Example usage

val email = Email("john.doe@example.com", "Meeting Reminder", "Don't
forget our meeting at 10 AM.")

val sms = SMS("123-456-7890", "Hey, are you available for a call?")

println(showNotification(email))

println(showNotification(sms))
```

Question 5

Quick sort not needed

Question 6

```
import scala.io.StdIn

object CapitalizeWords {

  def capitalizeEachWord(sentence: String): String = {

    sentence.split(" ").map(word => word.head.toUpper +
word.tail.toLowerCase).mkString(" ")

  }

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

    println("Enter a sentence:")

    val sentence = StdIn.readLine()

    val capitalizedSentence = capitalizeEachWord(sentence)

    println(s"Capitalized sentence: $capitalizedSentence")

  }

}
```

```
CapitalizeWords.main(Array())
Enter a sentence:
my name is don
Capitalized sentence: My Name Is Don
```

Question 7

```
object FunctionalQuickSort {
  // Define the quickSort function
  def quickSort(arr: List[Int]): List[Int] = arr match {
    case Nil => Nil // Base case: empty list
    case pivot :: tail =>
      val (left, right) = tail.partition(_ < pivot)
      quickSort(left) ::: pivot :: quickSort(right)
  }

  // Main function to test the implementation
  def main(args: Array[String]): Unit = {
    val arr = List(10, 7, 8, 9, 1, 5)
    println("Original array:")
    println(arr.mkString(", "))

    val sortedArr = quickSort(arr)
    println("Sorted array:")
    println(sortedArr.mkString(", "))
  }
}
```

Question 8:

```
import scala.collection.mutable.Map

object ItemCollection {
  // Define the mutable map to represent the collection of
items
  var items: Map[String, Int] = Map(
    "Pen" -> 20,
    "Pencil" -> 10,
    "Eraser" -> 7,
    "Book" -> 25,
    "Sheet" -> 15
  )

  def main(args: Array[String]): Unit = {
    // i. Display item-name and quantity
    println("Items in the collection:")
    items.foreach { case (item, quantity) =>
      println(s"$item: $quantity")
    }

    // ii. Display sum of quantity and total number of items
    val totalQuantity = items.values.sum
    val totalItems = items.size
    println(s"\nTotal Quantity: $totalQuantity")
    println(s"Total Number of Items: $totalItems")

    // iii. Add 3 Books to the collection
    items.update("Book", items.getOrElse("Book", 0) + 3)

    // Add new item "Board" with quantity 15 to the
collection
    items += ("Board" -> 15)

    // Display updated collection
    println("\nUpdated Items in the collection:")
    items.foreach { case (item, quantity) =>
      println(s"$item: $quantity")
    }
  }
}
```

(Or)

```
var items = scala.collection.mutable.Map("Pen"->2 , "Gun"->3
, "Scale"->4);
items += ("Apple"->3);
items.values.sum
```

Question 9:

```
object SearchElement {
  def search(numbers: List[Int], target: Int): Boolean = {
    // Using the contains method of List to check if the
target exists in the list
    numbers.contains(target)
  }

  def main(args: Array[String]): Unit = {
    // Example usage:
    val numbers = List(1, 2, 3, 4, 5, 6, 7)
    val target1 = 5
    val target2 = 10

    println(s"Searching for $target1 in the list:
${search(numbers, target1)}")
    println(s"Searching for $target2 in the list:
${search(numbers, target2)}")
  }
}
```

Question 10:

```
def counter(n: Int): Unit = {
     | for (i <- 0 to n){
     | println(i)
     | }
     | }
```

```
counter(10) // Calling function
```

Question 11:

```
def factorial(n: Int): Int = {
      if (n <= 1) 1
      else n * factorial(n - 1)
    }
var arr = Array(1,2,3,5);
arr.foreach(x=>println(factorial(x)))
```

Question 12:

```
var grocery =
scala.collection.mutable.Map("Butter"->20,"Bun"->10,"Biscuit
"->7,"Bread"->5)

grocery.foreach{case
(i,q)=>println("Item:"+i+",Quantity:"+q)}

scala> grocery.size // To get total number of items
res13: Int = 4
scala> grocery.values.sum //To get total quantity
res14: Int = 42

scala> grocery("Bun") += 5
scala> grocery("Bun") //Updating a map item
res17: Int = 15
```

Question 13

```
object BinarySearch {

  def binarySearch(list: List[Int], target: Int): Boolean = {
    def search(start: Int, end: Int): Boolean = {
      if (start > end) false
      else {
        val mid = (start + end) / 2
        if (list(mid) == target) true
        else if (list(mid) > target) search(start, mid - 1)
        else search(mid + 1, end)
      }
    }
    search(0, list.length - 1)
  }
  def main(args: Array[String]): Unit = {
    val sortedList = List(1, 3, 5, 7, 9, 11)
    val target = 7
    println(binarySearch(sortedList, target)) // Output: true
  }
}
```

Question 14

```
import scala.io.StdIn.readLine
object LongestWord {
  def findLongestWord(words: List[String]): (String, Int) = {
    words.map(word => (word, word.length)).maxBy(_._2)
  }
  def main(args: Array[String]): Unit = {

    println("Enter words separated by commas (e.g.,
games,television,rope,table):")

    val input = readLine()

    val words = input.split(",").map(_.trim).toList

    val (longestWord, length) = findLongestWord(words)

    println(s"The longest word is '$longestWord' with length
$length.")

  }

}
```

(OR)

```
var words = List("ANS","BN","POPi");
words: List[String] = List(ANS, BN, POPi)
scala> var x = words.reduce((x,y)=>{
     | if(x.length > y.length)
     | x
     | else
     | y
     | }
     | )
x: String = POPi
```

Question 15

Using aggregate()

```
val rdd = sc.parallelize(Seq(1, 2, 3, 4, 5))

val zeroValue = List[Int]()
val seqOp = (acc: List[Int], value: Int) => acc :+ (value + 100)

val combOp = (acc1: List[Int], acc2: List[Int]) => acc1 ++ acc2

val resultAggregate = rdd.aggregate(zeroValue)(seqOp, combOp)


println(resultAggregate) // Output: List(101, 102, 103, 104, 105)
```

Using fold()

For fold(), we need a different approach since fold() is designed for reducing the entire RDD to a single value, not for element-wise transformations. However, we can illustrate its use by summing the elements after adding 100 to each.

```
val zeroValueFold = 0

val foldOp = (acc: Int, value: Int) => acc + (value + 100)

val resultFold = rdd.fold(zeroValueFold)(foldOp)

println(resultFold) // Output: 515 (i.e., (1+100) + (2+100) + (3+100)
+ (4+100) + (5+100))
```

Question 16

```
val filePath = "C:\\Users\\opopopop\\Downloads\\WORDDD.txt"

val textFileRDD = sc.textFile(filePath)

val wordsRDD = textFileRDD.flatMap(line => line.split("\\W+"))

//My name      is  => Split (‘My’ , ‘name’ , ‘ ‘ , ‘ ‘,’is’)

val filteredWordsRDD = wordsRDD.filter(word => word.nonEmpty)

val lowerCaseWordsRDD = filteredWordsRDD.map(word => word.toLowerCase)
val wordCountPairsRDD = lowerCaseWordsRDD.map(word => (word, 1))

val wordCounts = wordCountPairsRDD.reduceByKey(_ + _)

wordCounts.take(10).foreach(println)

val frequentWords = wordCounts.filter { case (word, count) => count >
4 }

val outputFilePath = "C:\\Users\\opopop\\Downloads\\res.txt"

frequentWords.saveAsTextFile(outputFilePath)

frequentWords.collect().foreach(println)
```

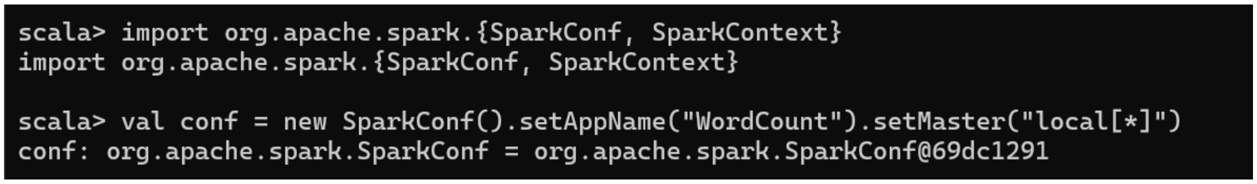
Question 17

```
import org.apache.spark.{SparkConf, SparkContext}

object WordCount {

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

    // Initialize Spark configuration

val conf = new
SparkConf().setAppName("WordCount").setMaster("local[*]")

    val sc = new SparkContext(conf)


    // Path to your input text file

    val filePath = "path/to/your/text.txt"

    // Read the text file into an RDD

    val textFileRDD = sc.textFile(filePath)


    // Split each line into words
    val wordsRDD = textFileRDD.flatMap(line => line.split("\\W+"))


    // Filter out any empty words

    val filteredWordsRDD = wordsRDD.filter(word => word.nonEmpty)


    // Convert each word to lowercase and map to (word, 1) pairs

 val wordPairsRDD = filteredWordsRDD.map(word => (word.toLowerCase,1))


    // Reduce by key to get word counts

    val wordCountsRDD = wordPairsRDD.reduceByKey(_ + _)


    // Collect and print the word counts

    wordCountsRDD.collect().foreach { case (word, count) =>

      println(s"$word: $count")

    }

    // Stop Spark context

    sc.stop()

  }

}
```

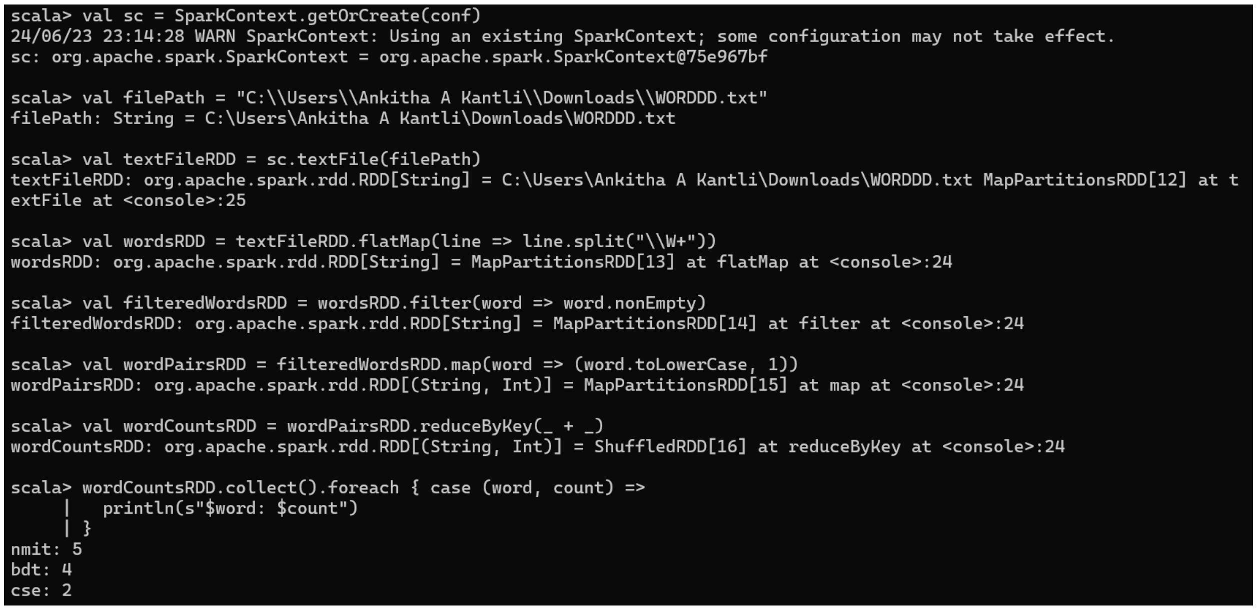
Output

```
scala> WordCount.main(Array(""))

nmit: 5

bdt: 4
```

cse: 2



```
scala> import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.{SparkConf, SparkContext}

scala> val conf = new SparkConf().setAppName("WordCount").setMaster("local[*]")
conf: org.apache.spark.SparkConf = org.apache.spark.SparkConf@69dc1291
```



```
scala> val sc = SparkContext.getOrCreate(conf)
24/06/23 23:14:28 WARN SparkContext: Using an existing SparkContext; some configuration may not take effect.
sc: org.apache.spark.SparkContext = org.apache.spark.SparkContext@75e967bf

scala> val filePath = "C:\\Users\\Ankitha A Kantli\\Downloads\\WORDDD.txt"
filePath: String = C:\Users\Ankitha A Kantli\Downloads\WORDDD.txt

scala> val textFileRDD = sc.textFile(filePath)
textFileRDD: org.apache.spark.rdd.RDD[String] = C:\Users\Ankitha A Kantli\Downloads\WORDDD.txt MapPartitionsRDD[12] at t
extFile at <console>:25

scala> val wordsRDD = textFileRDD.flatMap(line => line.split("\\W+"))
wordsRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[13] at flatMap at <console>:24

scala> val filteredWordsRDD = wordsRDD.filter(word => word.nonEmpty)
filteredWordsRDD: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[14] at filter at <console>:24

scala> val wordPairsRDD = filteredWordsRDD.map(word => (word.toLowerCase, 1))
wordPairsRDD: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[15] at map at <console>:24

scala> val wordCountsRDD = wordPairsRDD.reduceByKey(_ + _)
wordCountsRDD: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[16] at reduceByKey at <console>:24

scala> wordCountsRDD.collect().foreach { case (word, count) =>
     |   println(s"$word: $count")
     | }
nmit: 5
bdt: 4
cse: 2
```

Question 18

{"id":"572692378957430785",

"user":"Srkian_nishu smile",

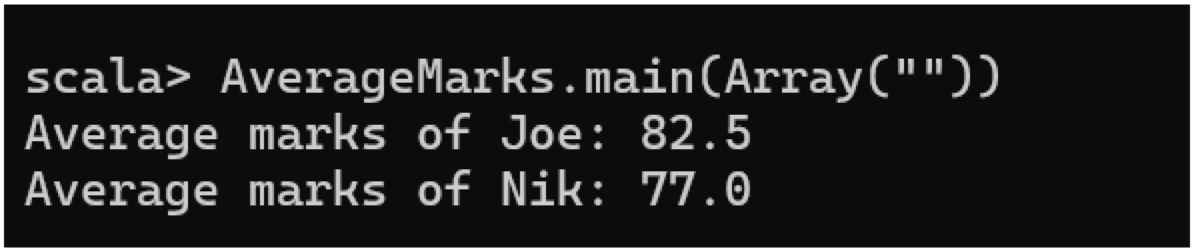
```
"text":"@always_nidhi @YouTube no idnt understand bti loved of
this mve is rocking",

"place":"Orissa",

"country":"India"}
```

```
import org.apache.spark.{SparkContext,SparkConf}


object tweetMining{

   val conf = new SparkConf().setAppName("User
Mining").setMaster("local[*]")


   val sc = new SparkContext(conf)


   var pathToFile = "/home/student/tweetmining/rt.json"


   def main(args:Array[String]){


   val tweets =
sc.textFile(pathToFile).mapPartitions(TweetUtils.parseFromJson(_
))

   //(John , “hello”) , (John , “hi”)
   val tweetsByUser = tweets.map(x=>(x.user,x)).groupByKey()

   //(John,(“hello” , “hi”))

   val numberOfTweets = tweetsByUser.map(x=>(x._1,x._2.size))

   //(John->2)

   val sorted = numberOfTweets.sortBy(_._2,ascending=false)


   sorted.take(10).foreach(println)

   }

}


import com.google.gson._


object TweetUtils {

   case class Tweet (

      id: String,


      user: String,


      userName: String,


      text: String,
        place: String,

        country: String,

        lang: String
    )

    def parseFromJson(lines:Iterator[String]):Iterator[Tweet] = {

        val gson = new Gson
        lines.map(line=>gson.fromJson(line,classOf[Tweet]))
    }
}
```

Question 20(Average Marks):

```
import org.apache.spark.{SparkConf, SparkContext}
object AverageMarks {
  def main(args: Array[String]): Unit = {
    val conf = new
SparkConf().setAppName("AverageMarks").setMaster("local[2]")
```

```
    val sc = new SparkContext(conf)

    val data = Array(

      ("Joe", "Maths", 83),

      ("Joe", "Physics", 74),

      ("Joe", "Chemistry", 91),

      ("Joe", "Biology", 82),

      ("Nik", "Maths", 69),

      ("Nik", "Physics", 62),

      ("Nik", "Chemistry", 97),

      ("Nik", "Biology", 80)

    )


    val rdd = sc.parallelize(data)

    // Step 1: Convert to PairRDD: (StudentName, (Marks, 1))

    val pairRDD = rdd.map { case (student, subject, marks) =>
(student, (marks, 1)) }



    // Step 2: Aggregate by key to get (StudentName, (TotalMarks,
Count))

    val totalMarksCount = pairRDD.reduceByKey { case ((marks1,
count1), (marks2, count2)) =>

      (marks1 + marks2, count1 + count2)

    }

    // Step 3: Calculate average marks
    val averageMarks = totalMarksCount.mapValues { case (totalMarks,
count) =>

      totalMarks / count.toDouble

    }

    // Step 4: Collect and print the result

    averageMarks.collect().foreach { case (student, avgMarks) =>

      println(s"Average marks of $student: $avgMarks")

    }

    sc.stop()

  }

}
```

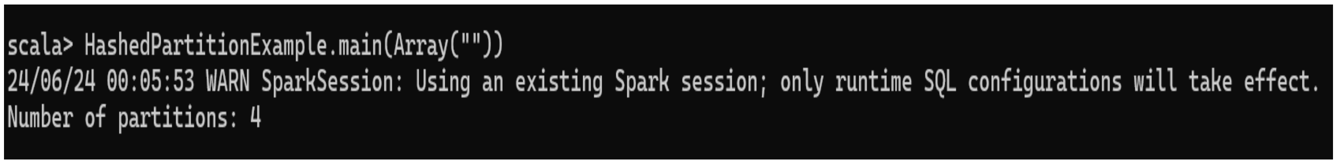


```
scala> AverageMarks.main(Array(""))
Average marks of Joe: 82.5
Average marks of Nik: 77.0
```

Question 21

```
import org.apache.spark.sql.{SparkSession, DataFrame}

object HashedPartitionExample {

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

    val spark = SparkSession.builder()

      .appName("HashedPartitionExample")
      .master("local[*]") // Replace with your Spark master URL if
running in a cluster

      .getOrCreate()


    // Sample data for illustration

    import spark.implicits._

    val data = Seq(

      (1, "HR", "Manager"),

      (2, "IT", "Developer"),

      (3, "HR", "Assistant"),

      (4, "Finance", "Accountant"),

      (5, "IT", "Analyst"),

      (6, "Finance", "Manager")

    )

    val employeeDF = data.toDF("EmpID", "Dept", "EmpDesg")


    // Hash partitioning by 'Dept' into 4 partitions

    val numPartitions = 4

    val hashedPartitionedDF = employeeDF.repartition(numPartitions,
$"Dept")


    // Verify number of partitions

    println(s"Number of partitions:
${hashedPartitionedDF.rdd.partitions.length}")
```

```
    // Perform further operations as needed on hashedPartitionedDF


    // Stop Spark session
    spark.stop()
  }
}
```

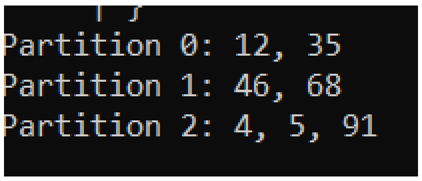


```
scala> HashedPartitionExample.main(Array(""))
24/06/24 00:05:53 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
Number of partitions: 4
```

Question 23(Partitions):

```
import org.apache.spark.{SparkConf, SparkContext}

val conf = new
SparkConf().setAppName("PartitionExample").setMaster("local[*]")

val sc = new SparkContext(conf)

val data = Seq(11, 34, 45, 67, 3, 4, 90)

val rdd = sc.parallelize(data, 3)

val result = rdd.mapPartitionsWithIndex((index, iter) => {
  iter.map(x => (index, x + 1))
}).groupBy(_._1)

result.collect().foreach { case (index, values) =>
```

```
  println(s"Partition $index: ${values.map(_._2).mkString(", ")}")

}
```



```
Partition 0: 12, 35
Partition 1: 46, 68
Partition 2: 4, 5, 91
```

(OR)

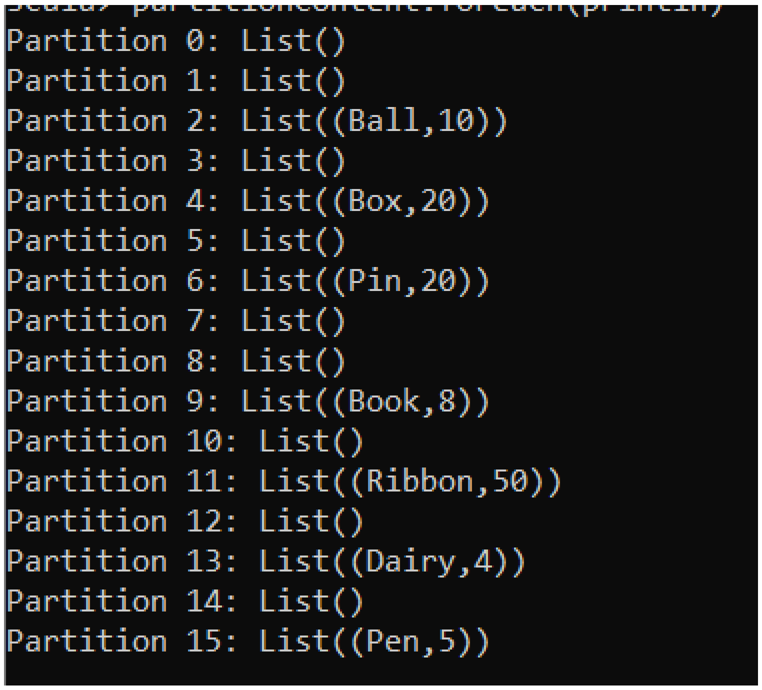
```
val rdd = sc.parallelize(Array(11,34,45,67,3,4,90),3)
 val mpp = rdd.mapPartitionsWithIndex{(pi , pitr) => {
     | println("partion:"+pi)
     | val mylist = pitr.toList
     | mylist.map(x =>{println(x);x+1}).iterator
     | }
     | }
mpp.collect
```

Question 24(Partitions2):

```
val conf = new
SparkConf().setAppName("ItemPartitioner").setMaster("local[*]")
val sc = new SparkContext(conf)

val item = Map("Ball" -> 10, "Ribbon" -> 50, "Box" -> 20, "Pen"
-> 5, "Book" -> 8, "Dairy" -> 4, "Pin" -> 20)

val rdd = sc.parallelize(item.toSeq)
val numPartitions = rdd.getNumPartitions println(s"Number of
partitions created for the collection Item: $numPartitions")
```

```
val partitionContent = rdd.mapPartitionsWithIndex { (index,
iter) => Iterator(s"Partition $index: ${iter.toList}")
}.collect()
```

```
partitionContent.foreach(println)
```



```
Partition 0: List()
Partition 1: List()
Partition 2: List((Ball,10))
Partition 3: List()
Partition 4: List((Box,20))
Partition 5: List()
Partition 6: List((Pin,20))
Partition 7: List()
Partition 8: List()
Partition 9: List((Book,8))
Partition 10: List()
Partition 11: List((Ribbon,50))
Partition 12: List()
Partition 13: List((Dairy,4))
Partition 14: List()
Partition 15: List((Pen,5))
```

Question 25(Partitions3):

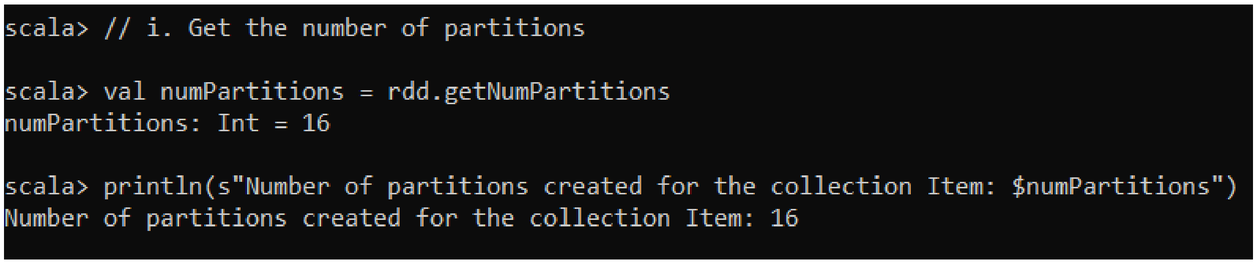
```
val conf = new
SparkConf().setAppName("ItemPartitioner").setMaster("local[*]")
```

```
val sc = new SparkContext(conf)
```

```
val item = Map("Ball" -> 10, "Ribbon" -> 50, "Box" -> 20, "Pen"
-> 5, "Book" -> 8, "Dairy" -> 4, "Pin" -> 20)
```

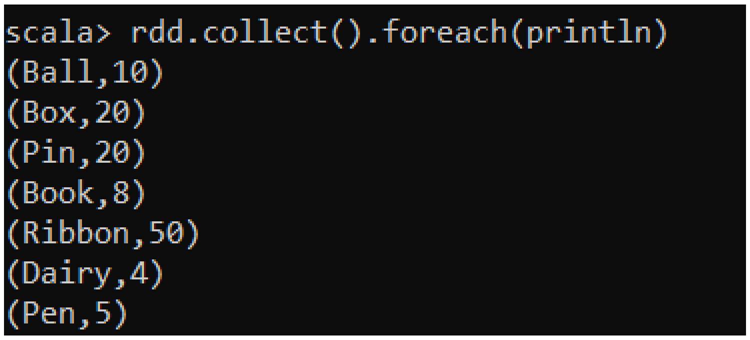
```
val rdd = sc.parallelize(item.toSeq)
```

```
// i. Get the number of partitions

val numPartitions = rdd.getNumPartitions

println(s"Number of partitions created for the collection Item:
$numPartitions")
```



```
scala> // i. Get the number of partitions

scala> val numPartitions = rdd.getNumPartitions
numPartitions: Int = 16

scala> println(s"Number of partitions created for the collection Item: $numPartitions")
Number of partitions created for the collection Item: 16
```

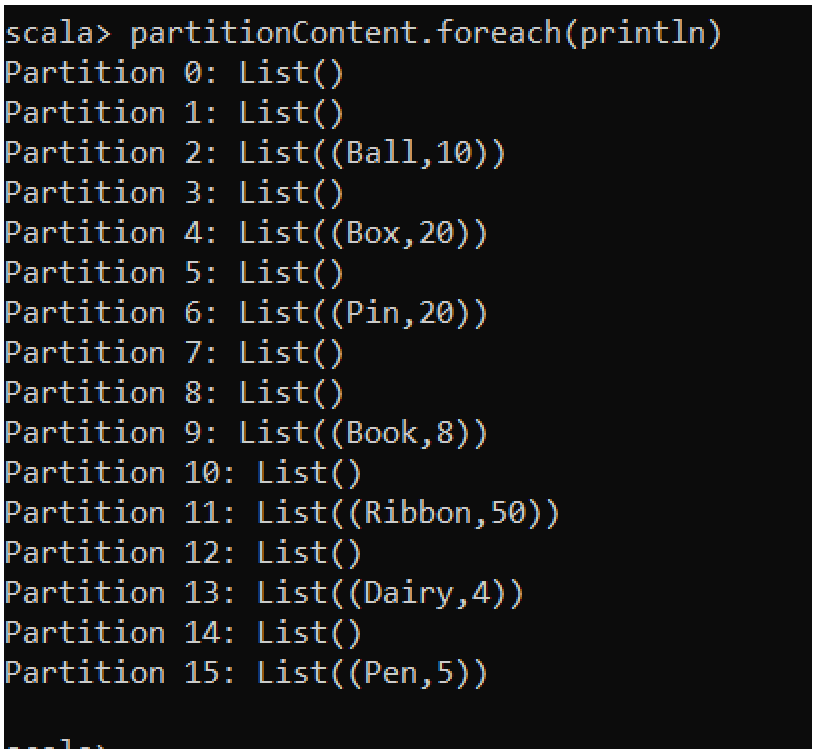
```
// ii. Display the content of the RDD

println("Content of the RDD:")

rdd.collect().foreach(println)
```



```
scala> rdd.collect().foreach(println)
(Ball,10)
(Box,20)
(Pin,20)
(Book,8)
(Ribbon,50)
(Dairy,4)
(Pen,5)
```

```
// iii. Display the content of each partition separately

val partitionContent = rdd.mapPartitionsWithIndex { (index,
iter) =>

  Iterator(s"Partition $index: ${iter.toList}")

}.collect()

// Print the content of each partition
```

```
println("\nContent of each partition separately:")

partitionContent.foreach(println)
```



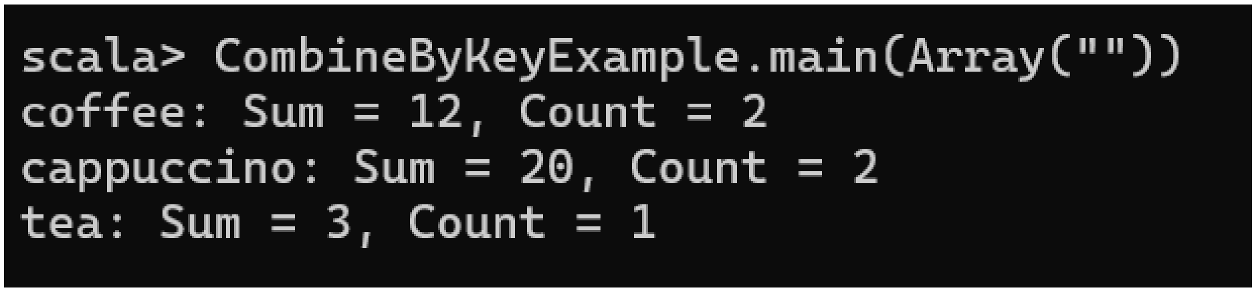
```
scala> partitionContent.foreach(println)
Partition 0: List()
Partition 1: List()
Partition 2: List((Ball,10))
Partition 3: List()
Partition 4: List((Box,20))
Partition 5: List()
Partition 6: List((Pin,20))
Partition 7: List()
Partition 8: List()
Partition 9: List((Book,8))
Partition 10: List()
Partition 11: List((Ribbon,50))
Partition 12: List()
Partition 13: List((Dairy,4))
Partition 14: List()
Partition 15: List((Pen,5))
```

Question 26:

```
 var ldd =
sc.textFile("/Users/shreyasgs/Desktop/Scala/ex.txt");
var c = ldd.flatMap(line=>line.split("\\W+"))
var oc = c.map(x => (x , 1));
 var red = oc.reduceByKey(_ + _);
red.collect.sortBy(_._1)
var stws = red.filter{ case(word,_)
=>{word.startsWith("S")|| word.startsWith("s")}}
```

Question 27

```
import org.apache.spark.{SparkConf, SparkContext}

object CombineByKeyExample {

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

    val conf = new
SparkConf().setAppName("CombineByKeyExample").setMaster("local[*
]")

    val sc = new SparkContext(conf)



    // Input data

    val data = Seq(("coffee", 2), ("cappuccino", 5), ("tea", 3),
("coffee", 10), ("cappuccino", 15))



    // Create an RDD from the data

    val rdd = sc.parallelize(data)



    // Apply combineByKey

    val combinedRDD = rdd.combineByKey(

      // Create combiner: Initialize the accumulator (sum,
count) for each key

      (value: Int) => (value, 1),
      // Merge value: Incorporate a new value (quantity) into
the accumulator

      (acc: (Int, Int), value: Int) => (acc._1 + value, acc._2 +
1),



      // Merge combiners: Merge accumulators from different
partitions

      (acc1: (Int, Int), acc2: (Int, Int)) => (acc1._1 +
acc2._1, acc1._2 + acc2._2)

    )



    // Print the result

    combinedRDD.collect().foreach { case (key, (sum, count)) =>

      println(s"$key: Sum = $sum, Count = $count")

    }



    sc.stop()

  }

}
```



```
scala> CombineByKeyExample.main(Array(""))
coffee: Sum = 12, Count = 2
cappuccino: Sum = 20, Count = 2
tea: Sum = 3, Count = 1
```