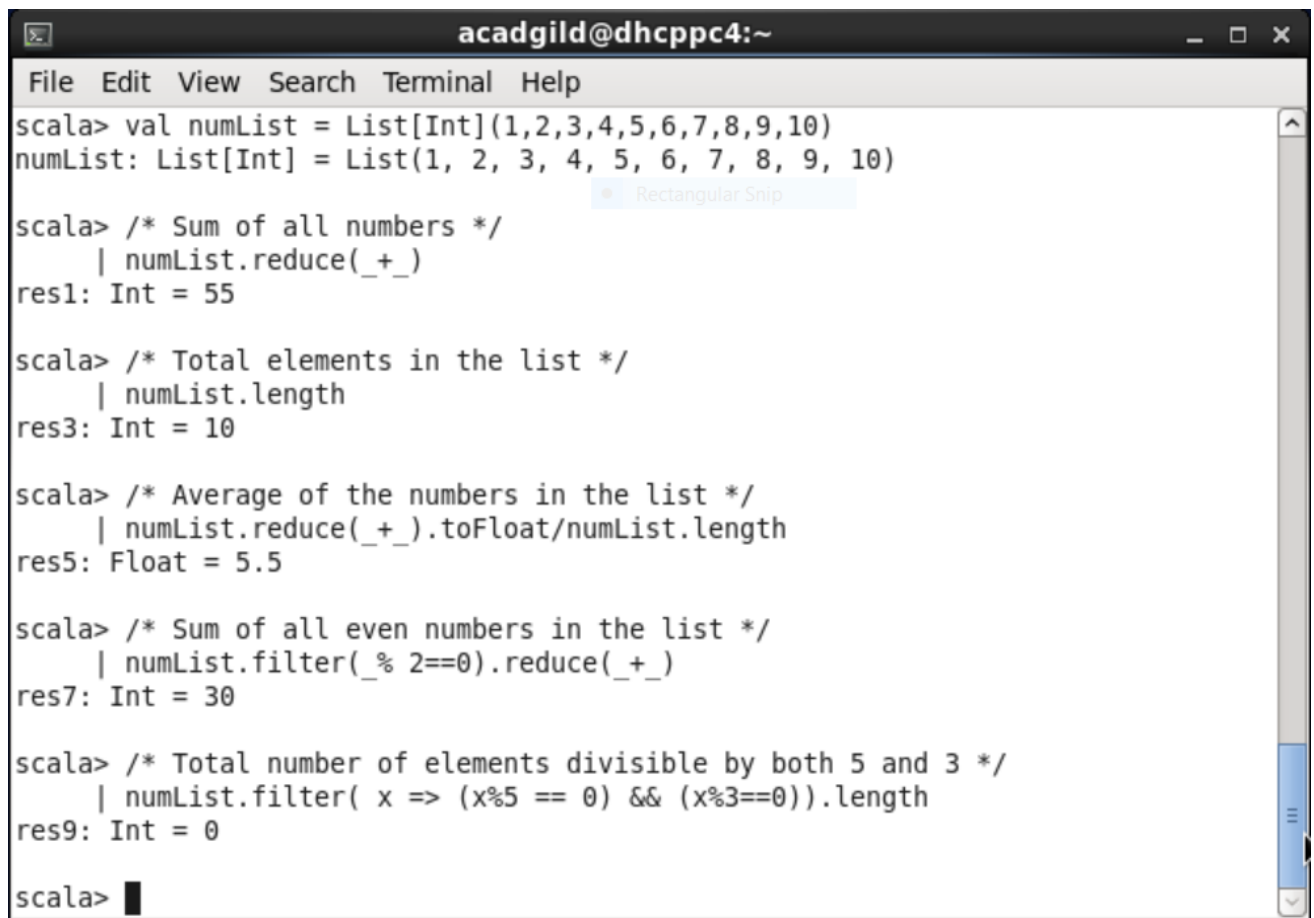


Task 1:



```
acadgild@dhcppc4:~  
File Edit View Search Terminal Help  
scala> val numList = List[Int](1,2,3,4,5,6,7,8,9,10)  
numList: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
  
scala> /* Sum of all numbers */  
      | numList.reduce(_+_)  
res1: Int = 55  
  
scala> /* Total elements in the list */  
      | numList.length  
res3: Int = 10  
  
scala> /* Average of the numbers in the list */  
      | numList.reduce(_+_).toFloat/numList.length  
res5: Float = 5.5  
  
scala> /* Sum of all even numbers in the list */  
      | numList.filter(_%2==0).reduce(_+_)  
res7: Int = 30  
  
scala> /* Total number of elements divisible by both 5 and 3 */  
      | numList.filter( x => (x%5 == 0) && (x%3==0)).length  
res9: Int = 0  
  
scala> █
```

Task 2:

1. Pen down the limitations of MapReduce.

Limitations of MapReduce:

- Cannot handle interactive processing
- Cannot perform real-time (stream) processing
- Cannot do iterative (delta) processing
- Cannot do in-memory processing or cacheing
- Cannot perform graph processing
- It has high latency which may be unsuitable in various applications
- It is not easy to use since it requires complex code for each and every operation
- Most often cannot handle complex machine-learning type algorithms
- With too many keys, sorting may take a very long time

2. What is RDD? Explain few features of RDD?

RDD stands for “Resilient Distributed Dataset” and is a fundamental data structure of Apache Spark.

RDD Features:

- In-memory computation: stores intermediate results in RAM instead of disk
- Lazy evaluations: do not compute right away, but keep track of required transformations and compute only at the right time
- Fault tolerance: track data lineage and rebuild lost data automatically upon failure
- Immutability: data is safe to share across processes
- Partitioning: each partition is a logical division of data
- Persistence: user can state which RDD they would want to reuse
- Coarse-grained operations: achieved by map or filter or group-by operations
- Location stickiness: can define placement preference to compute partitions

3. List down few Spark RDD operations and explain each of them.

RDD in Apache Spark supports 2 types of operations –

- a. Transformation,**
- b. Action**

RDD Transformations are functions that take an RDD as input and produce one or many RDDs as output via lazy operations.

- ❖ **Narrow Transformations:**
Data is from a single partition, and is result of map, filter, Flatmap, MapPartition, Sample and Union functions

❖ **Wide or Shuffle Transformations:**

Data may live in many partitions of the parent RDD and use functions such as Intersection, Distinct, ReduceByKey, GroupByKey, Join, Cartesian, Repartition and Coalesce

RDD Actions returns final result of RDD computations.

- ❖ It triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.
- ❖ Actions produce non-RDD values.

A few Spark RDD operations:

Transformations:

Map:

Map will take each row as input and return an RDD for the row.

Flat map:

flatMap will take an iterable data as input and returns the RDD as the contents of the iterator.

Filter:

filter returns an RDD which meets the filter condition. Below is the sample demonstration of the above scenario.

ReduceByKey:

reduceByKey takes a pair of key and value pairs and combines all the values for each unique key. Below is the sample demonstration of the above scenario.

Actions:

Collect:

collect is used to return all the elements in the RDD. Refer the below screen shot for the same.

Count:

count is used to return the number of elements in the RDD.

CountByValue:

countByValue is used to count the number of occurrences of the elements in the RDD.

Take:

take will display the number of records we explicitly specify.

Task 3 :

Part 1 & 2:

```
scala> /* ----- Task 3 ----- */
| /* Part1 - Write a program to read a text file and print the number of rows of data in the document */
| val lines = sc.textFile("C:\\sometext.txt") // Read text file
lines: org.apache.spark.rdd.RDD[String] = C:\\sometext.txt MapPartitionsRDD[3] at textFile at <console>:26

scala> lines.foreach(println) // View the text
The CentOS Project is a community-driven free software effort focused on delivering a robust open source ecosystem.
We're expanding the availability of CentOS images to many of vendors, providing official images for Amazon, Google, and more.
For users, we offer a consistent manageable platform that suits a wide variety of deployments.
For self-hosted cloud, we also provide a generic cloud-init enabled image.
We offer a solid, predictable base to build upon, along with extensive resources to build, test, release, and maintain code.
The program works if it shows 6 rows and 93 words

scala> lines.count() // View number of lines in the text
res8: Long = 6

scala> /* ----- Task 3 ----- */
| /* Part2 - Write a program to read a text file and print the number of words in the document */
| val words = lines.flatMap(line => line.split(" ")) // Split by Space Character
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at flatMap at <console>:27

scala> words.count() // View the word count of the text
res11: Long = 93

scala>
```

Part 3:

```
scala> /* ----- Task 3 ----- */
| /* Part 3 - We have a document where the word separator is -, instead of space. Write a spark code, to obtain the
| count of the total number of words present in the document */
| val lines = sc.textFile("C:\\sampledoc.txt") // Read the sample document text file
lines: org.apache.spark.rdd.RDD[String] = C:\\sampledoc.txt MapPartitionsRDD[9] at textFile at <console>:26

scala> val words = lines.flatMap(line => line.split("-")) // Split by - character
words: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[10] at flatMap at <console>:25

scala> words.count() // Total word count
res12: Long = 22

scala> lines.count() // Total line count in the Sample Document
res13: Long = 3

scala>
```