Assignment 18.1

Task 1

Given a list of numbers - List[Int] (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
- find the sum of all numbers
- find the total elements in the list
- calculate the average of the numbers in the list
- find the sum of all the even numbers in the list
- find the total number of elements in the list divisible by both 5 and 3

Solution#1

Create the RDD and initialize with the List of number 1 to 10

```
scala> val rdd = sc.parallelize(Seq(1,2,3,4,5,6,7,8,9,10))
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[12] at parallelize
at <console>:24
```

Print the sum of the elements of the RDD

```
scala> rdd.sum
res25: Double = 55.0
```

Print the number of elements of the RDD

```
scala> rdd.count
res26: Long = 10
```

Calculate the average of the numbers (sum/count)

```
scala> rdd.sum/rdd.count
res27: Double = 5.5
```

Sum of all even numbers of the list. We use the filter to keep only even numbers and then find sum for it

```
scala> rdd.filter(x=>(x%2)==0).sum
res34: Double = 30.0
```

To find the number of elements in the list divisible by both 3 and 5

```
scala> rdd.filter(x=>((x%3)==0) && ((x%5)==0)).count
res44: Long = 0
```

Task 2

1) Pen down the limitations of MapReduce.
2) What is RDD? Explain few features of RDD?
3) List down few Spark RDD operations and explain each of them.

Solution#2

Limitations of Map Reduce

Performance – Map reduce uses disk IO extensively during the map and reduce phases. The data is sorted and shuffled in the reduce phase which uses lot of network bandwidth

Batch processing – Since there is a lot latency in Map reduce computation , it can only be used for batch processing. Map reduce is not suitable for streaming data as well

Lack of Delta Iteration – Map reduce does not support iterative processing in itself and multiple map reduce jobs need to be sequenced by an external program like oozie. This makes is unsuitable for iterative algorithms

Not easy to use – Map reduce requires developers to write lot of code for simple operations and so it is not easy to use compared to Spark etc. Map reduce does not offer any high level of abstraction for ease of development

Lack of optimization - Map reduce framework does not provide any kind of optimization when various map reduce jobs are linked and run together.

What is RDD

Spark RDD is a resilient, partitioned, distributed and immutable collection of Data. It is a basic construct in Spark for data processing. RDD hold the data in the form of Scala collection. Following are the properties of the RDD

Resilient - They can recover from failure. RDDs are fault tolerant which means RDD can be recreated if a particular job fails

Partitioned - Spark breaks the RDD in smaller chunks of data. These pieces are called partitions.

Distributed - Instead of keeping these partitions on a single machine, spark spreads them across the cluster for parallel processing of data

Immutability - Spark RDD is immutable/read only data structure. It can be created in only two ways

- Load data from a source,
- by transforming another RDD

RDD Operations

RDD offers two type of operations Transformations and Actions

Transformation operation creates a new RDD from an existing RDD. Typical example include filtering, joining or groupBy operations. All transformations are lazy until an action requires them to provide

Actions are mainly performed to send results back to the driver and hence they produce a non distributed dataset. For example Collect is an action which gets the result back to the driver

Below are a list of some common operations on RDD

- **Aggregate** - Aggregate the elements of each partition, and then the results for all the partitions
- **Count** - Return the number of elements in the RDD.
- **Distinct** - Return a new RDD containing the distinct elements in this RDD
- **groupBy** - Return an RDD of grouped items.
- **Map** - Return a new RDD by applying a function to all elements of this RDD
- **Collect** - Return an RDD that contains all matching values by applying f
- **Filter** - Return a new RDD containing only the elements that satisfy a predicate