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

    val numlist = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

    val rawrdd = sc.parallelize(numlist)

    rawrdd.sum()

- find the total elements in the list

    rawrdd.count()

- calculate the average of the numbers in the list

    rawrdd.sum()/rawrdd.count()

- find the sum of all the even numbers in the list

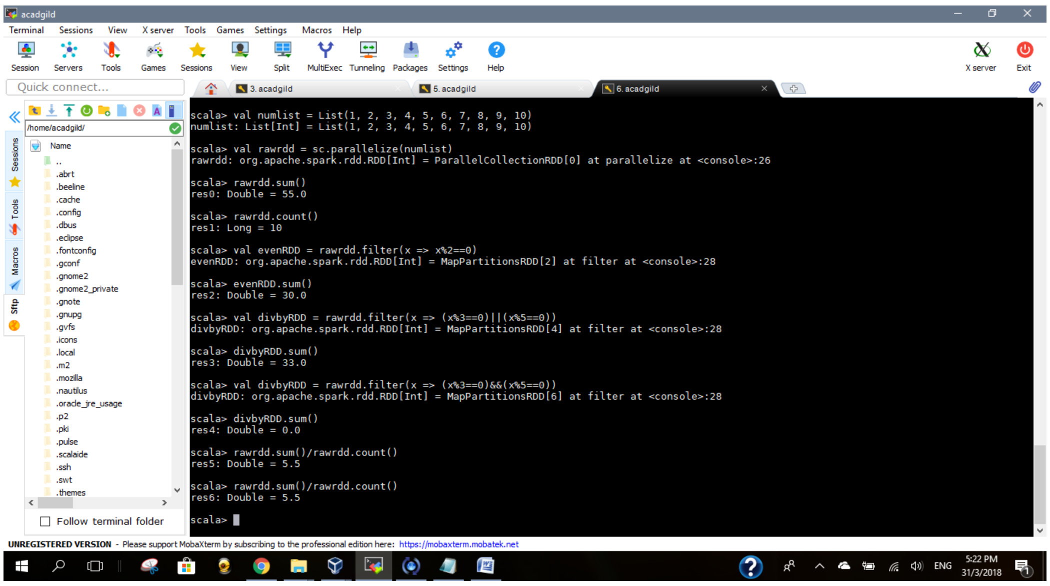
    val evenRDD = rawrdd.filter(x => x%2==0)

    evenRDD.sum()

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

    val divbyRDD = rawrdd.filter(x => (x%3==0)&&(x%5==0))

    divbyRDD.sum()



```
scala> val numlist = List(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> val rawrdd = sc.parallelize(numlist)
rawrdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:26

scala> rawrdd.sum()
res0: Double = 55.0

scala> rawrdd.count()
res1: Long = 10

scala> val evenRDD = rawrdd.filter(x => x%2==0)
evenRDD: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[2] at filter at <console>:28

scala> evenRDD.sum()
res2: Double = 30.0

scala> val divbyRDD = rawrdd.filter(x => (x%3==0)||(x%5==0))
divbyRDD: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[4] at filter at <console>:28

scala> divbyRDD.sum()
res3: Double = 33.0

scala> val divbyRDD = rawrdd.filter(x => (x%3==0)&&(x%5==0))
divbyRDD: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[6] at filter at <console>:28

scala> divbyRDD.sum()
res4: Double = 0.0

scala> rawrdd.sum()/rawrdd.count()
res5: Double = 5.5

scala> rawrdd.sum()/rawrdd.count()
res6: Double = 5.5

scala> 
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

Task 2

1) Pen down the limitations of MapReduce.
    a. Hadoop MR is not suitable for small data. It lacks the ability to support random reading of small files because it is designed for high capacity files.
    b. Data is distributed and processed over cluster in MR which increases the process time and reduces the process speed with its parallel and distributed algorithm.
    c. MR mainly supports batch processing. Not for real time streamed data processing, and hence overall performance is slower. MR does not leverage memory of Hadoop cluster to the maximum.
    d. Hadoop MR is not efficient for iterative process as it does not support cyclic data flow, that is chain of stages which each output of previous become input of next process.
    e. MR takes huge data in one format and translates to another format data structure. Individual elements are **break down in key,Value pair** and Reduce takes output from Map. This process take lot of time perform Reduce process.
    f. MR intermediate data processing **cannot be cache** for further requirements which dismisses the performance.

2) What is RDD? Explain few features of RDD?
    - Resilient Distributed Dataset.
    - Resilient means it can be recomputed in case loose the data/part of data.
    - It is a logical reference of dataset which is partitioned across cluster. RDD are immutable and self recovered in case of failure.
    - RDD is the fundamental data structure of SPARK. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

    Features of RDD

    - In memory computation – It stores intermediate results in distributed memory (RAM) instead of stable storage.
    - Lazy evaluation – meaning when we call some operation in RDD, it does not execute immediately. Hence, Lazy evaluation data is not loaded until it is necessary.
    - Fault Tolerance - RDD remembers how it was created from other datasets to create by itself.
    - Immutability – Data is safe to share across process. It can also be created and retrieved any time which makes caching, sharing and replication easily.
    - Partitioning – Each partition is one logical division of data which is mutable.

3) List down few Spark RDD operations and explain each of them.
    a. Transformations
        i. RDD transformations are functions that make RDD as the input and produce one or many RDDs as the output by applying computations like map(), filter(), reducedbykey(), etc. They do not change input RDD as it is immutable.

    ii. Transformations are lazy operations since it requires to trigger a function to operate transformations and produce new data sets.
    iii. Certain transformations can be pipelined which is an optimized methods to improve performance. Below are the two kinds of transformations.
        1. Narrow / Pipelining - is the result of map, filter and such data is from single partition only. ie, it is self sufficient.
        2. Wide / Shuffle – is the result of groupByKey, reduceByKey like functions. The data required to compute from single partition may live in may many partitions of parent RDD.

b. Action
    i. An action in spark 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 result to driver or to file.
    ii. First(), take(), reduce(), collect(), count() are some of Actions in spark to produce non RDD values.