Vijay's Assignment – Spark 1

Task 1

Given a list of numbers - List[Int] (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) – list1 defined as given below

- find the sum of all numbers – Used list1.sum

- find the total elements in the list – Used list1.length

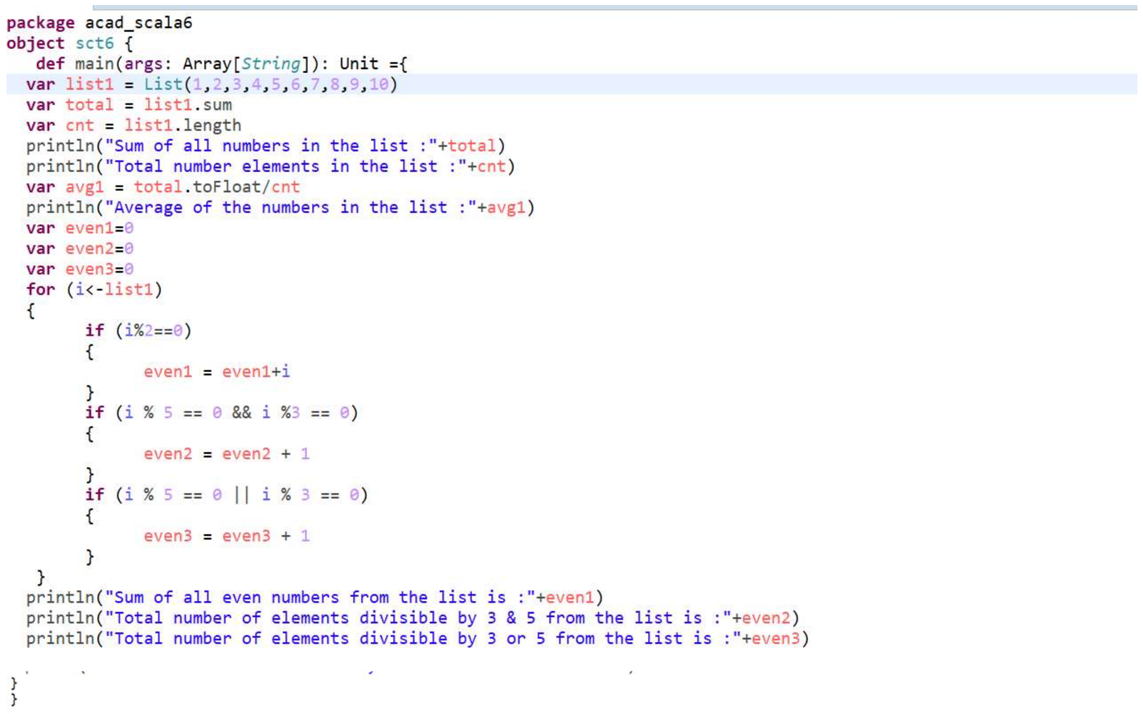
- calculate the average of the numbers in the list – Divided list1.sum by list1.length and converted the output to float

- find the sum of all the even numbers in the list – Used for loop to identify numbers divisible by 2 in the list

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

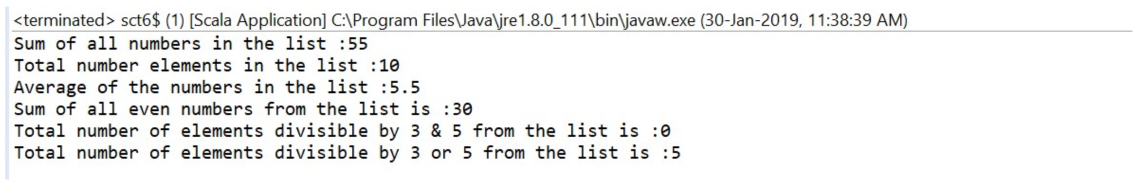
    - Used the for loop to identify numbers divisible by both 5 & 3

    - Used the for loop to identify numbers divisible by eithe 5 or 3



```
package acad_scala6
object sct6 {
   def main(args: Array[String]): Unit ={
  var list1 = List(1,2,3,4,5,6,7,8,9,10)
  var total = list1.sum
  var cnt = list1.length
  println("Sum of all numbers in the list :"+total)
  println("Total number elements in the list :"+cnt)
  var avg1 = total.toFloat/cnt
  println("Average of the numbers in the list :"+avg1)
  var even1=0
  var even2=0
  var even3=0
  for (i<-list1)
  {
        if (i%2==0)
        {
              even1 = even1+i
        }
        if (i % 5 == 0 && i %3 == 0)
        {
              even2 = even2 + 1
        }
        if (i % 5 == 0 || i % 3 == 0)
        {
              even3 = even3 + 1
        }
   }
  println("Sum of all even numbers from the list is :"+even1)
  println("Total number of elements divisible by 3 & 5 from the list is :"+even2)
  println("Total number of elements divisible by 3 or 5 from the list is :"+even3)

}
}
```

Output



```
<terminated> sct6$ (1) [Scala Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (30-Jan-2019, 11:38:39 AM)
Sum of all numbers in the list :55
Total number elements in the list :10
Average of the numbers in the list :5.5
Sum of all even numbers from the list is :30
Total number of elements divisible by 3 & 5 from the list is :0
Total number of elements divisible by 3 or 5 from the list is :5
```

Task 2

1) Pen down the limitations of MapReduce.

   Map reduce is the original data processing framework developed for Hadoop. Map reduce has following limitations

   - High Latency – Map reduce is suitable only for batch processing and not for online processing
   - Excessive Disk I/O – The data nodes where the mapper executes, the result of the mapper is stored within that nodes local disk. Every mapper operation reads data from the disk and stores the output back in the disk. There is no caching of intermediate results in local memory (RAM)
   - Excessive Network I/O – To perform the reducer operation, mapper output from all data notes will be sent to the data node executing the reducer. This causes excessive network read and write operations
   - Even though Pig & Hive were developed to provide wrappers for complex Map reduce code, they still invoke the Map reduce code in the backend. Hence the output from Pig & Hive has a high latency compared to Spark
   - Map Reduce is not suitable for Real time data processing. It cannot handle streaming data coming from live sources

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

   - RDD stands for Resilient Distributed datasets. RDDs are the fundamental data structure for Spark. It is an immutable distributed collection of objects. Each RDD is divided into Partitions and each partition can be stored across different nodes of the cluster.
   - Features of RDD are
     - Read only partitioned collection of records
     - Fault tolerant collection of elements
     - Can be operated in parallel
     - Supports in memory computation ideal for data sharing
     - In Spark RDD, intermediate results are stored in memory for faster processing instead of disk
     - RDD can be persisted in memory for faster access by multiple queries

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

   There are two types of operations on Spark RDD

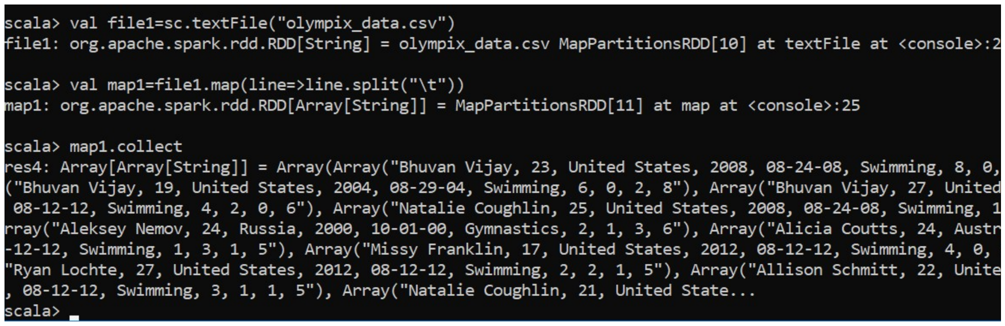
   - Transformation
   - Action

   Transformation

   Transformation operation takes one RDD as input and creates one ore more RDDs as output. Each time a transformation is applied to a RDD a new RDD is created. Transformations are lazzy as they are not executed immediately. They are executed only when an action is called.

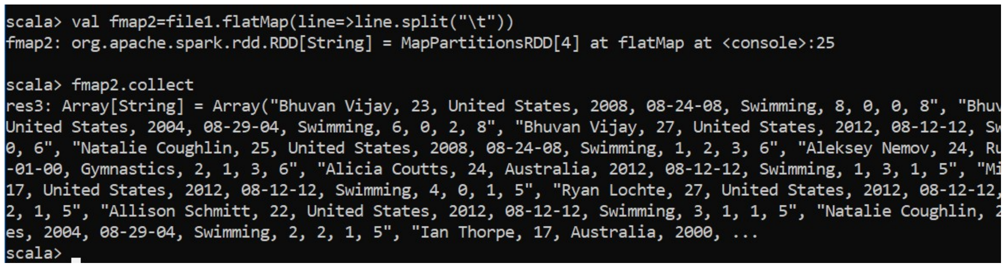
Few Examples of transformation:

1) map
   a. applies the input function to every element of RDD. Output is iteratable



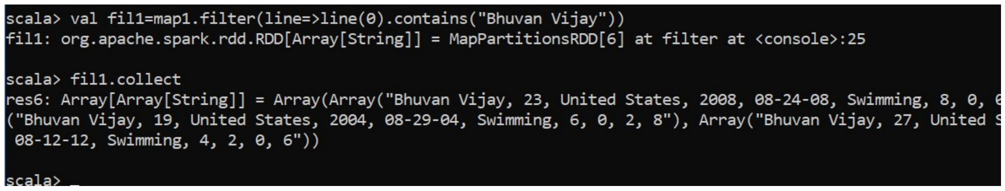
```
scala> val file1=sc.textFile("olympix_data.csv")
file1: org.apache.spark.rdd.RDD[String] = olympix_data.csv MapPartitionsRDD[10] at textFile at <console>:2

scala> val map1=file1.map(line=>line.split("\t"))
map1: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[11] at map at <console>:25

scala> map1.collect
res4: Array[Array[String]] = Array(Array("Bhuvan Vijay, 23, United States, 2008, 08-24-08, Swimming, 8, 0,
("Bhuvan Vijay, 19, United States, 2004, 08-29-04, Swimming, 6, 0, 2, 8"), Array("Bhuvan Vijay, 27, United
 08-12-12, Swimming, 4, 2, 0, 6"), Array("Natalie Coughlin, 25, United States, 2008, 08-24-08, Swimming, 1
rray("Aleksey Nemov, 24, Russia, 2000, 10-01-00, Gymnastics, 2, 1, 3, 6"), Array("Alicia Coutts, 24, Austr
-12-12, Swimming, 1, 3, 1, 5"), Array("Missy Franklin, 17, United States, 2012, 08-12-12, Swimming, 4, 0,
"Ryan Lochte, 27, United States, 2012, 08-12-12, Swimming, 2, 2, 1, 5"), Array("Allison Schmitt, 22, Unite
, 08-12-12, Swimming, 3, 1, 1, 5"), Array("Natalie Coughlin, 21, United State...
scala> _
```

2) flatMap
   a. Applies the input function to every element of RDD and produces multiple elements in output RDD. Output is non iteratable.



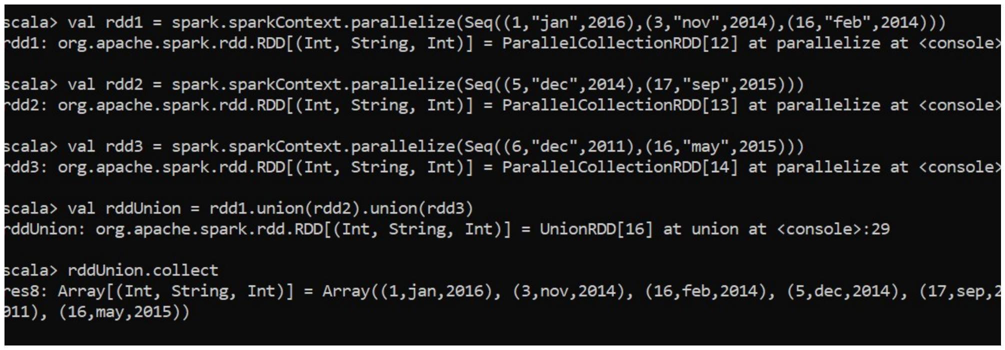
```
scala> val fmap2=file1.flatMap(line=>line.split("\t"))
fmap2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at flatMap at <console>:25

scala> fmap2.collect
res3: Array[String] = Array("Bhuvan Vijay, 23, United States, 2008, 08-24-08, Swimming, 8, 0, 0, 8", "Bhuv
United States, 2004, 08-29-04, Swimming, 6, 0, 2, 8", "Bhuvan Vijay, 27, United States, 2012, 08-12-12, Sw
0, 6", "Natalie Coughlin, 25, United States, 2008, 08-24-08, Swimming, 1, 2, 3, 6", "Aleksey Nemov, 24, Ru
-01-00, Gymnastics, 2, 1, 3, 6", "Alicia Coutts, 24, Australia, 2012, 08-12-12, Swimming, 1, 3, 1, 5", "Mi
17, United States, 2012, 08-12-12, Swimming, 4, 0, 1, 5", "Ryan Lochte, 27, United States, 2012, 08-12-12,
2, 1, 5", "Allison Schmitt, 22, United States, 2012, 08-12-12, Swimming, 3, 1, 1, 5", "Natalie Coughlin, 2
es, 2004, 08-29-04, Swimming, 2, 2, 1, 5", "Ian Thorpe, 17, Australia, 2000, ...
scala> _
```

3) filter
   a. Returns a new RDD with only the elements that satisfy the predicate mentioned



```
scala> val fil1=map1.filter(line=>line(0).contains("Bhuvan Vijay"))
fil1: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[6] at filter at <console>:25

scala> fil1.collect
res6: Array[Array[String]] = Array(Array("Bhuvan Vijay, 23, United States, 2008, 08-24-08, Swimming, 8, 0, 0
("Bhuvan Vijay, 19, United States, 2004, 08-29-04, Swimming, 6, 0, 2, 8"), Array("Bhuvan Vijay, 27, United S
 08-12-12, Swimming, 4, 2, 0, 6"))

scala> _
```

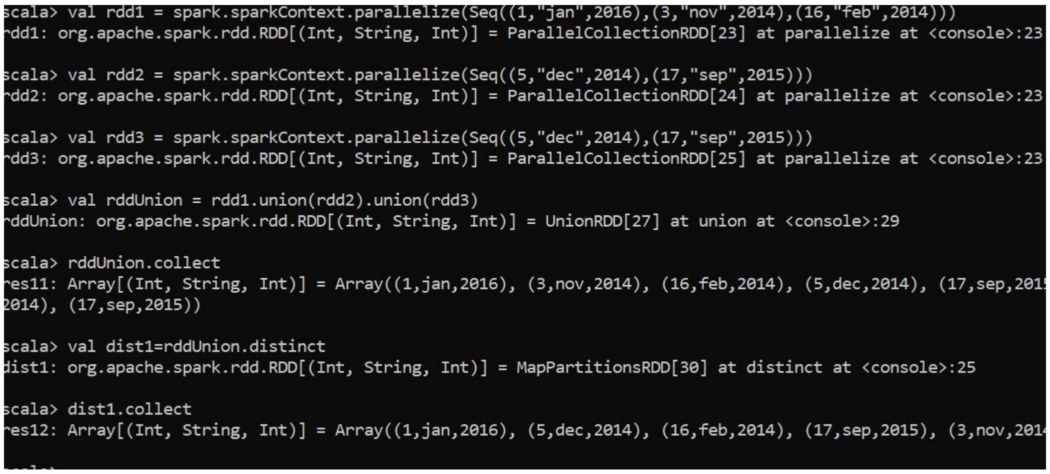
4) union
   a. Returns a new RDD that contains elements from both the input RDDs. Both RDDs should be of the same type



```
scala> val rdd1 = spark.sparkContext.parallelize(Seq((1,"jan",2016),(3,"nov",2014),(16,"feb",2014)))
rdd1: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[12] at parallelize at <console>

scala> val rdd2 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(17,"sep",2015)))
rdd2: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[13] at parallelize at <console>

scala> val rdd3 = spark.sparkContext.parallelize(Seq((6,"dec",2011),(16,"may",2015)))
rdd3: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[14] at parallelize at <console>

scala> val rddUnion = rdd1.union(rdd2).union(rdd3)
rddUnion: org.apache.spark.rdd.RDD[(Int, String, Int)] = UnionRDD[16] at union at <console>:29

scala> rddUnion.collect
res8: Array[(Int, String, Int)] = Array((1,jan,2016), (3,nov,2014), (16,feb,2014), (5,dec,2014), (17,sep,2
011), (16,may,2015))
```

5) distinct

   a. Returns a new RDD that contains the distinct elements of the source RDD



```
scala> val rdd1 = spark.sparkContext.parallelize(Seq((1,"jan",2016),(3,"nov",2014),(16,"feb",2014)))
rdd1: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[23] at parallelize at <console>:23

scala> val rdd2 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(17,"sep",2015)))
rdd2: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[24] at parallelize at <console>:23

scala> val rdd3 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(17,"sep",2015)))
rdd3: org.apache.spark.rdd.RDD[(Int, String, Int)] = ParallelCollectionRDD[25] at parallelize at <console>:23

scala> val rddUnion = rdd1.union(rdd2).union(rdd3)
rddUnion: org.apache.spark.rdd.RDD[(Int, String, Int)] = UnionRDD[27] at union at <console>:29

scala> rddUnion.collect
res11: Array[(Int, String, Int)] = Array((1,jan,2016), (3,nov,2014), (16,feb,2014), (5,dec,2014), (17,sep,201
2014), (17,sep,2015))

scala> val dist1=rddUnion.distinct
dist1: org.apache.spark.rdd.RDD[(Int, String, Int)] = MapPartitionsRDD[30] at distinct at <console>:25

scala> dist1.collect
res12: Array[(Int, String, Int)] = Array((1,jan,2016), (5,dec,2014), (16,feb,2014), (17,sep,2015), (3,nov,201
```

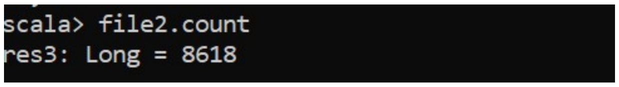
Action

Actions work on top of an existing RDD without producing any other RDDs as output. Actions are Spark RDD operations that produces non-RDD values as output.

Few examples of actions

1) Count

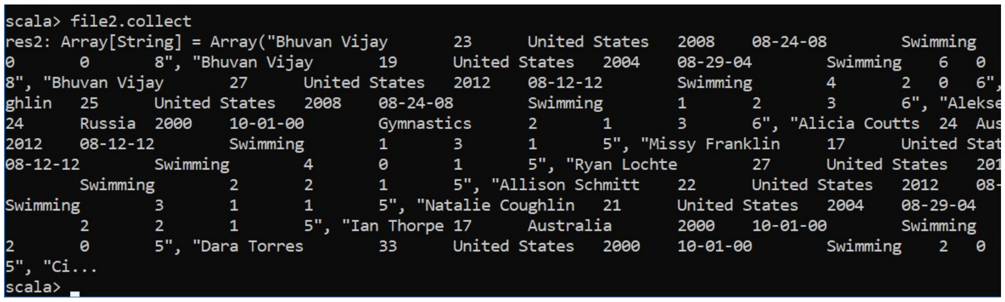
   a. Returns the number of elements in a RDD



```
scala> file2.count
res3: Long = 8618
```

2) Collect

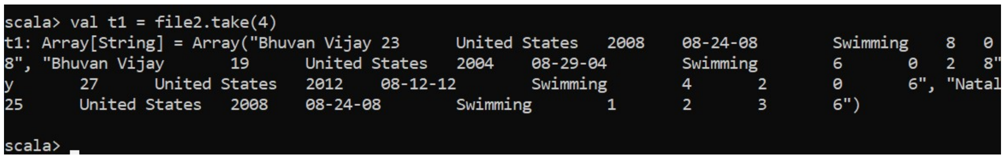
   a. Returns entire RDD to our driver program. Collect is used to display the contents of the RDD in console



```
scala> file2.collect
res2: Array[String] = Array("Bhuvan Vijay       23      United States   2008    08-24-08        Swimming
0       0       8", "Bhuvan Vijay       19      United States   2004    08-29-04        Swimming    6   0
8", "Bhuvan Vijay       27      United States   2012    08-12-12        Swimming        4       2   0   6",
ghlin   25      United States   2008    08-24-08        Swimming        1       2       3       6", "Alekse
24      Russia  2000    10-01-00        Gymnastics      2       1       3       6", "Alicia Coutts  24  Aus
2012    08-12-12        Swimming        1       3       1       5", "Missy Franklin     17      United Stat
08-12-12        Swimming        4       0       1       5", "Ryan Lochte        27      United States   201
        Swimming        2       2       1       5", "Allison Schmitt    22      United States   2012    08-
Swimming        3       1       1       5", "Natalie Coughlin   21      United States   2004    08-29-04
        2       2       1       5", "Ian Thorpe 17      Australia       2000    10-01-00        Swimming
2       0       5", "Dara Torres        33      United States   2000    10-01-00        Swimming    2   0
5", "Ci...
scala> 
```

3) Take

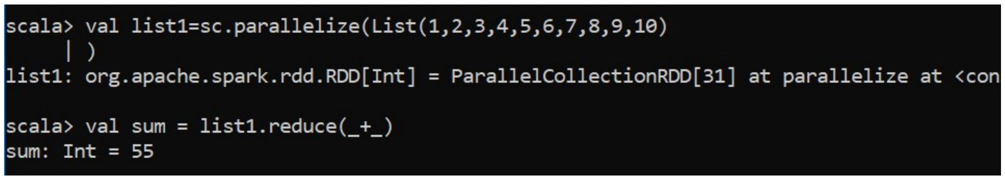
   a. Returns the number of elements from RDD. It tries to cut the number of partition it accesses, so it represents a biased collection. We cannot presume the order of the elements.



```
scala> val t1 = file2.take(4)
t1: Array[String] = Array("Bhuvan Vijay 23      United States   2008    08-24-08        Swimming    8   0
8", "Bhuvan Vijay       19      United States   2004    08-29-04        Swimming        6       0   2   8"
y       27      United States   2012    08-12-12        Swimming        4       2       0       6", "Natal
25      United States   2008    08-24-08        Swimming        1       2       3       6")

scala>
```

4) Reduce

   a. takes the two elements as input from the RDD and then produces the output of the same type as that of the input elements.



```
scala> val list1=sc.parallelize(List(1,2,3,4,5,6,7,8,9,10)
     | )
list1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[31] at parallelize at <con

scala> val sum = list1.reduce(_+_)
sum: Int = 55
```