

Girish Rajani
A20503736
CSP-554 Big Data Technologies
Homework 6

After successfully connecting to the master node via SSH command, the TestDataGen.class from assignment four was uploaded to the /home/hadoop directory via scp.

```
girish@LAPTOP-18TD7ESB MINGW64 ~  
$ scp -i d:/users/giris/downloads/emr-key-pair.pem d:/users/giris/downloads/Test  
DataGen.class hadoop@ec2-54-160-49-148.compute-1.amazonaws.com:/home/hadoop  
TestDataGen.class  
100% 2189 103.9KB/s 00:00
```

Figure 1 - Showing TestDataGen.class uploaded to /home/hadoop directory using scp

Exercise 1)

```
[hadoop@ip-172-31-2-46 ~]$ java TestDataGen  
Magic Number = 14955
```

Figure 2 - Showing the file TestDataGen.class executed using "java TestDataGen" and generating a magic number, 14955

```
[hadoop@ip-172-31-2-46 ~]$ hadoop fs -put /home/hadoop/foodratings14955.txt /user/  
hadoop
```

Figure 3 - Using -put command to copy foodratings14955.txt file from the /home/hadoop directory to HDFS /user/hadoop directory

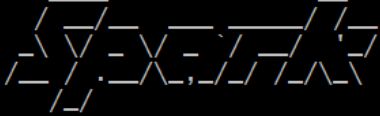
```
hadoop@ip-172-31-2-46 pydemo]$ pyspark  
Python 3.7.10 (default, Jun 3 2021, 00:02:01)  
[GCC 7.3.1 20180712 (Red Hat 7.3.1-13)] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(  
newLevel).  
22/10/05 21:40:50 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is s  
et, falling back to uploading libraries under SPARK_HOME.  
22/10/05 21:41:15 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to re  
quest executors before the AM has registered!  
Welcome to  
 version 2.4.8-amzn-2  
Using Python version 3.7.10 (default, Jun 3 2021 00:02:01)  
SparkSession available as 'spark'.
```

Figure 4 - Running PySpark

```
>>> ex1RDD = sc.textFile("/user/hadoop/foodratings14955.txt")
>>> ex1RDD.take(5)
['Me1,23,6,1,16,1', 'Me1,37,36,29,50,4', 'Sam,25,29,21,6,2', 'Jill,14,41,1,6,2', 'Jill,10,1,9,19,4']
```

Figure 5 - Using `sc.textFile` to read the `foodratings14955.txt` file into an RDD and then display the first five records using `take(5)`

Commands Used:

```
ex1RDD = sc.textFile("/user/hadoop/foodratings14955.txt")
```

```
ex1RDD.take(5)
```

Magic Number: **14955**

Exercise 2)

```
>>> ex2RDD = ex1RDD.map(lambda line: line.split(','))
>>> ex2RDD.take(5)
[['Me1', '23', '6', '1', '16', '1'], ['Me1', '37', '36', '29', '50', '4'], ['Sam', '25', '29', '21', '6', '2'], ['Jill', '14', '41', '1', '6', '2'], ['Jill', '10', '1', '9', '19', '4']]
```

Figure 6 - Showing the `split` function used to return a list of words to have a record being a list of items. The first five records were displayed using `take(5)`

Commands Used:

```
ex2RDD = ex1RDD.map(lambda line: line.split(','))
```

```
ex2RDD.take(5)
```

Exercise 3)

```
>>> ex3RDD = ex2RDD.map(lambda line : [line[0], line[1], int(line[2]), line[3], line[4], line[5]])
>>> ex3RDD.take(5)
[['Me1', '23', 6, '1', '16', '1'], ['Me1', '37', 36, '29', '50', '4'], ['Sam', '25', 29, '21', '6', '2'], ['Jill', '14', 41, '1', '6', '2'], ['Jill', '10', 1, '9', '19', '4']]
```

Figure 7 - Showing the lambda function `'lambda line : [line[0], line[1], int(line[2]), line[3], line[4], line[5]]'` used to convert the third column of each record from a string to an integer. The first five records were displayed using `take(5)`

Commands Used:

```
ex3RDD = ex2RDD.map(lambda line : [line[0], line[1], int(line[2]), line[3], line[4], line[5]])
```

```
ex3RDD.take(5)
```

Exercise 4)

```
>>> ex4RDD = ex3RDD.filter(lambda line : line[2]<25)
>>> ex4RDD.take(5)
[['Mel', '23', 6, '1', '16', '1'], ['Jill', '10', 1, '9', '19', '4'], ['Joy', '46',
10, '15', '15', '1'], ['Jill', '28', 5, '44', '3', '2'], ['Joe', '19', 21, '34',
'15', '5']]
```

Figure 8 - Showing a new RDD with a filter where the 3rd field in each record is less than 25 and the first five records were displayed using take(5)

Commands Used:

```
ex4RDD = ex3RDD.filter(lambda line : line[2]<25)
```

```
ex4RDD.take(5)
```

Exercise 5)

```
>>> ex5RDD = ex4RDD.map(lambda line: (line[0], line))
>>> ex5RDD.take(5)
[('Mel', ['Mel', '23', 6, '1', '16', '1']), ('Jill', ['Jill', '10', 1, '9', '19',
4']), ('Joy', ['Joy', '46', 10, '15', '15', '1']), ('Jill', ['Jill', '28', 5, '44',
3', '2']), ('Joe', ['Joe', '19', 21, '34', '15', '5'])]
```

Figure 9 - Showing a new RDD where each record is a key value pair where the key is the first field of the record and the value is the entire record. This was done using map transformation and then the first five records were displayed using take(5)

Commands Used:

```
ex5RDD = ex4RDD.map(lambda line: (line[0], line))
```

```
ex5RDD.take(5)
```

Exercise 6)

```
>>> ex6RDD = ex5RDD.sortByKey(True)
>>> ex6RDD.take(5)
[('Jill', ['Jill', '10', 1, '9', '19', '4']), ('Jill', ['Jill', '28', 5, '44', '3',
2']), ('Jill', ['Jill', '12', 8, '24', '41', '3']), ('Jill', ['Jill', '49', 9,
44', '1', '5']), ('Jill', ['Jill', '38', 23, '35', '40', '2'])]
```

Figure 10 - Showing a new RDD using sortByKey transformation to return the dataset of (K, V) pairs sorted by keys in ascending order and then the first five records were displayed using take(5)

Commands Used:

```
ex6RDD = ex5RDD.sortByKey(True)
```

```
ex6RDD.take(5)
```