Execises annotated with a (*) are optional.

Exercise 01:

Given the files Customer.txt and Order.txt, code the following SQL queries in Spark (Python and Scala).

1. SELECT name FROM Customer WHERE month(startDate)=7
2. SELECT DISTINCT name FROM Customer WHERE month(startDate)=7
3. SELECT O.cid, SUM(total), COUNT(DISTINCT total) FROM Order O GROUP BY O.cid
4. SELECT C.cid, O.total FROM Customer C, Order O WHERE month(startDate)=7 and C.cid=O.cid

Exercise 02:

Given the file urlaverage.txt, design and implement the Spark code (Python) to get the average consultation time of each url page in the file.

Exercise 03:

Download the file from the following link http://www.gutenberg.org/files/1342/1342-0.txt, then design and implement the Word Count algorithm in Spark (Python).

Once wordcount implemented, complete the algorithm by counting the frequency of each word in the document.

Please consider cleaning up the file by removing special characters like punctuation ones before the count.

To download the file:

- Windows command: using powershell: wget http://www.gutenberg.org/files/1342/1342-0.txt -OutFile wordcount.txt
- Linux command: wget http://www.gutenberg.org/files/1342/1342-0.txt

Exercise 04:

Graph analytics. Given the graph.txt file, design and implement algorithms in Spark for the following analytics problems on directed graphs. Assume that a graph is a set of edge pairs (v,w) indicating that an edge exists from node v to node w.

Please take into account that each line of the graph.txt file is of the form «v edge-label w w», so in order to represent a couple «v w» only the first and third elements should be retained for the sink calculation.

- **Sinks:** Given a directed graph, find the set of nodes having only incoming edges and having no outgoing edges.
- **Universal sinks:** Given a directed graph, find the set of nodes having an incoming edge from all the remaining nodes, and having no outgoing edges.

In order to first test out your algorithm, you can create a small RDD representing a graph via sc.parallelize(). E.g.: graph = sc.parallelize([('1','4'), ('2','4'), ('3','4'), ('3','2')])

Exercise 05: *

Design and implement in Spark (Python) a program to randomly generate two RDDs F1 and F2 including points/vectors in R5. For instance, a point in the files is represented as '3.45, 5.0, 0, 0, 2.0'.

Consider that F1.txt should contain at least 5 thousand points, while F2.txt must contain exactly 10 points.

Complete your program with a logic that allows you to compute the cross-product of the two RDDs without the use of the Spark cross-product operator. To illustrate this, refer to the following e.g.: if F1.txt contains '3.45, 5.0, 0, 0, 2.0' and F2.txt contains F1.txt contains '2.45,1.0, 2.0, 2.0', then the output of the job must contain their concatenation '3.45, 5.0, 0, 0, 2.0 2.45, 1.0, 2.0, 2.0'.

Tip : You can use the zipWithIndex transformation to add a unique index to each element of an RDD. For instance :

```
>>> rdd1 = sc.parallelize(['a','d','x'])
>>> rddzipped = rdd1.zipWithIndex()
>>> rddzipped.collect()
[('a', 0), ('d', 1), ('x', 2)]
```

Exercise 06: *

Do you think it is possible to perform an inner-join between two RDDs without using the join() transformation. If so, find the algorithm and code it into Spark. Just use the key-value RDDs used in classes to test your code :

- visits = sc.parallelize([("h", "1.2.3.4"), ("a", "3.4.5.6"), ("h","1.3.3.1")])
- pageNames = sc.parallelize([("h", "Home"), ("a", "About"), ("o", "Other")])

Does the same hold for right-outer join? If so, proceed as before.