**Exercise 0: Setting up the environment**

-Nothing to submit here-

a) Spark is available in Cloudera’s CDH and can be used out-of-the-box: open a terminal and type `/usr/bin/spark-shell` to start the Spark shell. Note though, that it may be slow to run (depending on your machine’s performance); as an alternative (and in contrast to Hadoop) Spark can also be installed quite simply following the next steps:

1. **Download the latest pre-built version from here:**
   
   http://spark.apache.org/downloads.html

2. **Unzip and move Spark to directory:**
   
   `cd ~/Downloads/`
   
   `tar xzvf spark-2.2.0-bin-hadoop2.7.tgz mv spark-2.2.0-bin-hadoop2.7/ spark`
   
   `sudo mv spark/ /usr/lib/`

3. **Make sure Java is installed. If not install Java.**

   `sudo apt-add-repository ppa:webupd8team/java`
   
   `sudo apt-get update`
   
   `sudo apt-get install oracle-java8-installer`

4. **Install SBT**

   `echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list`
   
   `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823`
   
   `sudo apt-get update`
   
   `sudo apt-get install sbt`
5. **Configure Spark.**
   
   ```
   cd /usr/lib/spark/conf/
   cp spark-env.sh.template spark-env.sh
   → Open spark-env.sh and add the following lines:
   JAVA_HOME=/usr/lib/jvm/java-8-oracle
   SPARK_WORKER_MEMORY=4g
   ```

6. **Configure bashrc**

   ```
   export JAVA_HOME=/usr/lib/jvm/java-8-oracle
   export SBT_HOME=/usr/share/sbt-launcher-packaging/bin/sbt-launch.jar
   export SPARK_HOME=/usr/lib/spark
   export PATH=$PATH:$JAVA_HOME/bin
   export PATH=$PATH:$SBT_HOME/bin:$SPARK_HOME/bin:$SPARK_HOME/sbin
   ```

7. **Now you are ready to interact with Spark:**

   type /usr/lib/spark-shell and start spark-shell


b) In this section we are going to write simple Scala functions to get used to the language.

1. Write a function that returns the second to last element of a list of integers.
   e.g. list=(1,2,3,4,5,6) → returns : 5
2. Write the same function but now use `zipWithIndex` and `filter` to obtain the same result.
3. Eliminate consecutive duplicates of a list elements. If a list contains repeated elements they should be replaced with a single copy of the element. The element order should not be changed.
   e.g. list=(1,2,3,3,5,5,8,10) → returns : (1,2,3,5,8,10)
4. Find the most frequent substring of size k on a list of strings. Assume k=4 and that all strings have length greater than k.
Exercise 1: Explore Web Logs with Spark (60%)

In this exercise you are going to explore the page views of Wikimedia projects. Download the page view statistics generated between 0-1am on Jan 1, 2016 from here:  ## [http://bit.ly/2q5WBDa](http://bit.ly/2q5WBDa) ##

Each line, delimited by a space, contains the statistics for one Wikimedia page.

The schema looks as follows:

<table>
<thead>
<tr>
<th>field</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>project code</td>
<td>the project identifier for each page</td>
</tr>
<tr>
<td>page title</td>
<td>a string containing the title of the page</td>
</tr>
<tr>
<td>num hits</td>
<td>number of requests on the specific hour</td>
</tr>
<tr>
<td>page size</td>
<td>size of the page</td>
</tr>
</tbody>
</table>

→ Launch the Spark shell and then create an RDD named `pagecounts` from the input file (the file must be copied on the same directory as the spark-shell):

```scala
val pagecounts = sc.textFile("/pagecounts-20160101-000000")
```

For each of the questions below implement a Scala function that takes as input an RDD and prints requested values. You must also include all of those results in your report.

1. Retrieve the first $k$ records and beautify.
   Use the take() operation of an RDD to get the first $k$ records, with $k = 15$.
   The take() operation returns an array and Scala simply prints the array with each element separated by a comma. This is not easy to read.
   Make the output prettier by traversing the array to print each record on its own line. (3 points)

2. Determine the number of records the dataset has in total. (3 points)

3. Compute the min, max, and average page size. (4 points)
   **Hint:** Use Map and Reduce/ReduceByKey functions provided by the RDD api.
   See here: [https://spark.apache.org/examples.html](https://spark.apache.org/examples.html)
4. Determine the record with the largest page size. (4 points)
   If multiple records have the same size, list all of them.

5. Determine the record with the largest page size again. (6 points)
   But now, pick the most popular.

6. Determine the record with the largest page title. (4 points)
   If multiple titles have the same length, list all of them.

7. Use the results of question 3, and create a new RDD with the records that have greater page size from the average. (6 points)

   **Hint**: Use function **filters** provided by the RDD Api.
   See here: [https://spark.apache.org/docs/2.1.1/programming-guide.html](https://spark.apache.org/docs/2.1.1/programming-guide.html)

8. Pageviews per project: compute the total number of pageviews for each project (as the schema shows, the first field of each record contains the project code). (3 points)

9. Report the 10 most popular pageviews of all projects, sorted by the total number of hits. (5 points)

10. Determine the number of page titles that start with the article “The”. (5 points)
    How many of those page titles are not part of the English project (Pages that are part of the English project have “en” as first field)?

11. Determine the percentage of pages that have only received a single page view in this one hour of log data. (5 points)

12. Determine the number of unique terms appearing in the page titles. Note that in page titles, terms are delimited by “” instead of a whitespace.
    You can use any number of normalization steps (e.g. lowercasing, removal of non-alphanumeric characters). (6 points)

13. Determine the most frequently occurring page title term in this dataset. (6 points)
**Exercise 2:**  **Spark Pseudodistributed Execution. (40%)**

On this assignment you are going to experiment with the the -pseudo- distributed execution on Spark. Your have to change the default configurations in order to emulate a local cluster environment with more than one nodes.

Follow the examples of the given link to setup the cluster settings: [http://bit.ly/2xrphYx](http://bit.ly/2xrphYx)

Configure the number of slaves and memory/cores that you think is more suitable for your PC, and create two simple topologies:

- **Small cluster**: with small number of workers, number of cores and slaves. *(20 points)*
- **Big cluster**: with the maximum computation power that you can give. *(20 points)*

For each setup mentioned above, run again questions 3,5,6,7,12,13 of Ex.1, and compare the execution time on the two emulated clusters.

You are asked not only to report the execution times for the two topologies, but **compare and explain** the two setup configurations and answer the following questions:

1. How master and slaves interact with the Spark environment?
2. Does the number of slaves affect the execution time?
3. Suppose you have input files that are greater than 10 GB and you are given a machine(like a google cloud) with 16 virtual CPUs and 60 GB of memory.
   You have two options:
   - a. 1 master and 1 slave with the maximum memory/computation power.
   - b. 1 masters and \( n \) slaves with the maximum computation power and memory divided across slaves.

   What is the range of \( n \) on this setup? What would you choose?
   How would you compare the options above with your emulated topologies?
   Is it better or worse?
• **Hints:**
  Experiment with the next variables:
  - SPARK_WORKER_CORES
  - SPARK_WORKER_INSTANCES
  - SPARK_WORKER_MEMORY

• **Useful tools to understand better your cluster environment are:**
  - Master node web UI: localhost:8080
  - Spark web UI (while running job): localhost:4040
  - Spark history server web UI: localhost:18080

History server is useful to inspect past jobs. For example when using spark-submit instead of spark-shell.
More info on how to start and config history server here:
[https://spark.apache.org/docs/latest/monitoring.html](https://spark.apache.org/docs/latest/monitoring.html)

**Submission (deadline: 16/11/2018):**

Create a folder including all the .scala files you used and a PDF report for the answers.
Send an email to hy562@csd.uoc.gr *(not the mailing list!)* with subject:
**Assign2__StudentIDnumber.**