MapReduce Exercise

```
# Go to https://github.com/rolandmueller/mapreduce
# Fork the repository
#

# Open a terminal on your laptop
# login to seneca3 with the login information from Moodle

# Clone your forked repository to seneca3
#
git clone https://github.com/YOURUSERNAME/mapreduce.git

# go into the mapreduce folder (cd = change directory)
#
cd mapreduce

# List the content of the current directory
# and check if all three files are there (ls = list) (-l parameter gives more infos)
# (-h human readable file size, e.g. 5MB)
#
ls -lh

# Make the mapper script executable (chmod = change mode) (+x add executable rights)
#
chmod +x mapper.py

# look how the access rights have changed
#
ls -l

# The cat command is outputting the file
# on the standard input (screen) (cat = "concatenate")
#
cat minipurchases.txt

# Look at the mapper code
# Read it and try to understand what it does
# Anticipate what the output would be
#
vim mapper.py

# Execute the mapper on the minipurchases.txt file
# the | (pipe) command puts the output of the left command
# as an input the right command
# do you understand why this output was created?
#
cat minipurchases.txt | ./mapper.py

# Sort the output of the mapper with "sort"
# again we connect the output of the mapper with | (pipe) to the sort command
#
cat minipurchases.txt | ./mapper.py | sort

# Make the reducer script executable
#
# TODO:
```

chmod +x reducer.py

```
# Look at the reducer code
# Read it and try to understand what it does
#
vim reducer.py

# Pipe the sorted output of the mapper to the reducer
#
cat minipurchases.txt | ./mapper.py | sort | ./reducer.py

# save the output in a file "result.txt"
# > saves the standard output to a file
#
cat minipurchases.txt | ./mapper.py | sort | ./reducer.py > result.txt

# Look at the result
# You can close less with the key "q"
#
less result.txt

# We tested the mapper and reducer locally and it works. Splendid!
# Now we want to run them in a Hadoop cluster

# Check what is in your home directory in HDFS
#
hadoop fs -ls

# Create an input folder in HDFS
#
hadoop fs -mkdir inputdata

# Put the data to HDFS in the folder inputdata
#
hadoop fs -put minipurchases.txt inputdata

# Check if the file is there in HDFS.
#
hadoop fs -ls inputdata

# Copy the mapper and reducer to the HDFS home folder
# by omitting the destination folder in the -put command
# the files will be saved in the home folder of Hadoop HDFS
#
hadoop fs -put reducer.py
hadoop fs -put mapper.py

# Change the mode (chmod) to executable (+x) for the mapper and reducer in Hadoop HDFS
#
hadoop fs -chmod +x mapper.py
hadoop fs -chmod +x reducer.py

# Open in your browser the following URL
# http://seneca1.lehre.hwr-berlin.de:9870
# In the menu click on Utilities|Browse the file system
# Click on the "user" directory
# find you own username and click on the directory
# check the content of the directory
```

```
# YARN (Yet Another Resource Negotiator) will handles the cluster management
# and will start map and reduce tasks on different remote servers.
# Normally mappers and reducers are programmed in Java.
# With Hadoop Streaming we can use also a mapper and a reducer written in Python
# (or any other programming language like R or Bash)
# and process any HDFS data.
# The following command will start the MapReduce job (write everything in one line).
#
yarn jar /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-streaming.jar -input
inputdata/* -output outputdata -mapper mapper.py -file mapper.py -reducer reducer.py -file
reducer.py

# Open the jobtracker Website in a browser to check the status of the MapReduce jobs
# http://seneca1.lehre.hwr-berlin.de:8088
# There you could see the MapReduce Job that you just have started in the last command
#

# This will take a while...
# When finished check if the output folder in HDFS.
#
hadoop fs -ls outputdata

# The output should look like that:
# Found 33 items
# -rw-r--r--   3 roland student          0 2019-05-02 21:35 outputdata/_SUCCESS
# -rw-r--r--   3 roland student         14 2019-05-02 21:35 outputdata/part-00000
# -rw-r--r--   3 roland student          0 2019-05-02 21:35 outputdata/part-00001
# ...
# _SUCESS means everything was ok
# the files part-00000 - part-00031 are the output

# Display the file part-00000
#
hadoop fs -cat outputdata/part-00000

# Concatenate (cat) all output files and display the result
#
hadoop fs -cat outputdata/part-*


# Get the merged output to the local file system and rename it to output.txt
#
hadoop fs -getmerge outputdata/ output.txt

# Look at the output
# (q for quit)
#
less output.txt

# We tested the mapper and reducer in Hadoop and it works also. Marvelous!
# The same MapReduce code would work also on a Hadoop cluster with 10.000 nodes
# and 1000 Terabytes of data.

# Write down in a separate text file on your laptop
# the equivalent SQL command for this MapReduce Job
```

```
# In the following exercises, you will change the mapper and reducer.
# If you rerun the MapReduce task, Hadoop will not overwrite the output folder, if the folder
# already exists. Instead Hadoop will stop with an error message.
# (this is a feature, not a bug ☺).
# Either change the name of the output folder in the MapReduce command
# to e.g. -output outputdata1
# or delete (-rm for remove -r for recursive delete)
# the output folder in HDFS on the command line:
#
hadoop fs -rm -r outputdata

# you can scroll to past commands that you used with the arrow up key
# so you do not have to type in previous commands again
#
# Create a commit after each exercise and push your code to your GitHub repository
#
# Write down in the separate text file on your laptop
# the equivalent SQL command for each MapReduce Job
#
# Test each exercise at least locally with the pipe commands,
# and optionally with Hadoop Streaming on YARN
#
# Exercises:
# 1. Change the mapper so that you calculate the sum for each category (not payment type)
# 2. Change the reducer so that you calculate the total number (count) of
#    purchases for each category (not the sum of the sales)
# 3. Change the mapper so that an error is raised when there are not six elements in the tuple
# 4. Change the mapper so that you calculate the sum only for the categories "Computers"
#    , "Cameras" and "Video Games"
# 5. Change the reducer, so that only categories are shown, that have a total number (count)
#    of more than 114 purchases
# 6. Change the reducer so that you calculate the average sales per category
#
# Push your code to your GitHub repository
# Submit the URL of your repository for the Moodle exercise
# Upload in Moodle the text file of the equivalent 7 SQL command for each 7 MapReduce Jobs
```