

# Big Data & Data Mining

## Activity 1: Biases in society

Cristina Barrado - 28/02/2025

---



---

## Index

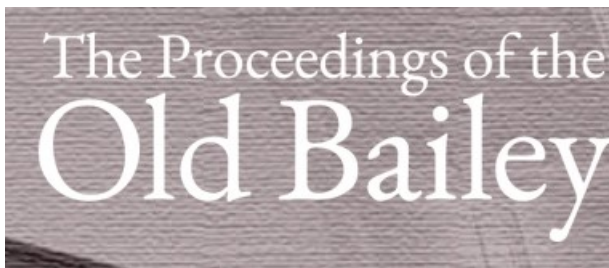
Introduction .....	3
Description of the problem.....	3
Input .....	3
Output .....	3
Document organization.....	4
Students' assignments (one practice per week).....	5
Week 1: frequency count in Python ( <i>Individual</i> ) .....	5
Week 2: frequency count with build-in libraries for Python ( <i>Pairs</i> ) .....	5
Week 3: frequency count in Spark ( <i>Pairs</i> ') .....	5
Guide of the Activity A1 in Python .....	6
Reading a local file.....	6
Passing parameters to a program.....	7
Execution time of a program.....	8
Installing new packages (Example in PyCharm) .....	9
Accessing to Internet documents and Parsing XML .....	10
Guide of the Activity A1 in Pandas .....	11
Reading a file .....	11
Guide of the Activity A1 in Spark .....	12
MapReduce in Pyhton (Sequential).....	12
Lambda, map, filter and reduce in Python .....	13
Data Distribution in SPARK (RDD).....	16
Map Reduce in SPARK (Parallel).....	17

---

## Introduction

This first activity is an introduction of three introductory programming tools to be used by all students during the first three weeks of the course. The three are guided programming tasks that will solve the same classical Big Data problem: Term Frequency (TF). The activity also addresses the horizontal competence about gender equality. The research to be solved here is the statistics about some old text, differentiating the treatment to men versus women. The first solution will be done in plain Python, the second solution will be created with libraries for Python and the third for Python/Spark.

## Description of the problem



The Proceedings of the Old Bailey contain 127 million words, recording 197,752 trials held at the Old Bailey, the Central Criminal Court in London between 1674 and 1913. You can browse into this link <https://www.oldbaileyonline.org/> to get some highlights and information about the content.

### Input

The access to the Internet web pages with Python can be done using the module **request** which takes an URL and returns the HTML of the requested page. The page has plain text, but also XML with tags. In the case of Old Bailey the returned pages contain also JavaScript code and shall be parsed with the **selenium** module instead. You could also access the data using the API (Application Program Interface). Look at the tutorial to understand better the functionalities, try several queries and notice the URL format and the returned data.

ALTERNATIVELY: Access the complete data already available in XML format from the University of Sheffield's data repository (ORDA):

<http://dx.doi.org/10.15131/shef.data.4775434>

The dataset consists of a ZIPPED file with 2,163 editions of the Proceedings and 475 Ordinary's Accounts marked up in TEI-XML, and contains some documentation covering the data structure and variables. Each Proceedings file represents one session of the court (1674-1913), and each Ordinary's Account file represents a single pamphlet (1676-1772). Look only at the court sessions documents. Further explanation on how the XML is organized can be found in these slides.

### Output

The output of the algorithm must list:

- the number of trials,
- the number of defendants,
- the number of them by gender,
- the number of convicted persons,

- 
- the number of them by gender
  - the ID of the ten most frequent convicted persons, and
  - the gender of each of these 10 most convicted persons.

Together with the numbers, your program shall generate a nice plots about the percent of convicted in global and by gender.

Output shall also include the execution time of the program and the average time per trial.

NOTE: In a trial a defendant can have several charges. The verdict is per person and per charge. To simplify the code, just consider that a defendant is declared guilty if one of the charges has a guilty verdict.

## Document organization

The rest of the document has the work to be done detailed by weeks. The functionality to implement is the one explained above, but you will test up to 6 different methods/tools, two each week, obtaining information about the time efficiency and scalability.

After this section, the document has a list of sections that present examples of code that you may find useful to implement your assignments.

---

## Students' assignments (one practice per week)

### Week 1: frequency count in Python (*Individual*)

Write a program in Python for the frequency count problem and test it with the 3 inputs (1 trail, 1 session -all trials of 1 day- and 1 month -30 sessions-).

Build 2 solutions to the problem, one for each of the following data structures to store the frequency counters:

- Using a plain list. Create lists from the XML, for instance a list [man, man, woman, ...] to hold the defendants gender, another with the convicted, another with the personID, ... and then write the code to obtain the counts.
- Using a dictionary. For instance using the gender or the personId as the key to access or modify the counter.

Check that the results are the same for the two versions of the code. Execute them with the three inputs and compare the execution time.

### Week 2: frequency count with build-in libraries for Python (*Pairs*)

Write a program in Python for the frequency count problem and run it with the 3 input datasets of first week. Build 2 solutions of the problem, one for each of the following build-in objects to store the frequency counters:

- Using a Counter object. Using the same lists of week-1-a, which contain repeated tokens, convert them to Counter objects and reduce the list to the pairs (name, counter).
- Using a Pandas Series and DataFrames. Read the XML file and create a Dataframe with 1 line per each defendant in each trial. Use function `value_counts()`.

Check that the results are the same for the two versions of the code. Execute them with the three files and compare the execution time.

### Week 3: frequency count in Spark (*Pairs*)

Write a program in Python for the Old Bailey dataset using the Map/Reduce paradigm and run it with the 3 input datasets of first week. Build 2 solutions to the problem:

- Using sequential Python functions map/filter/reduce
- Using parallel execution in Spark

Check that the results are the same and look for the execution time. Compare them also with results on Week 1.

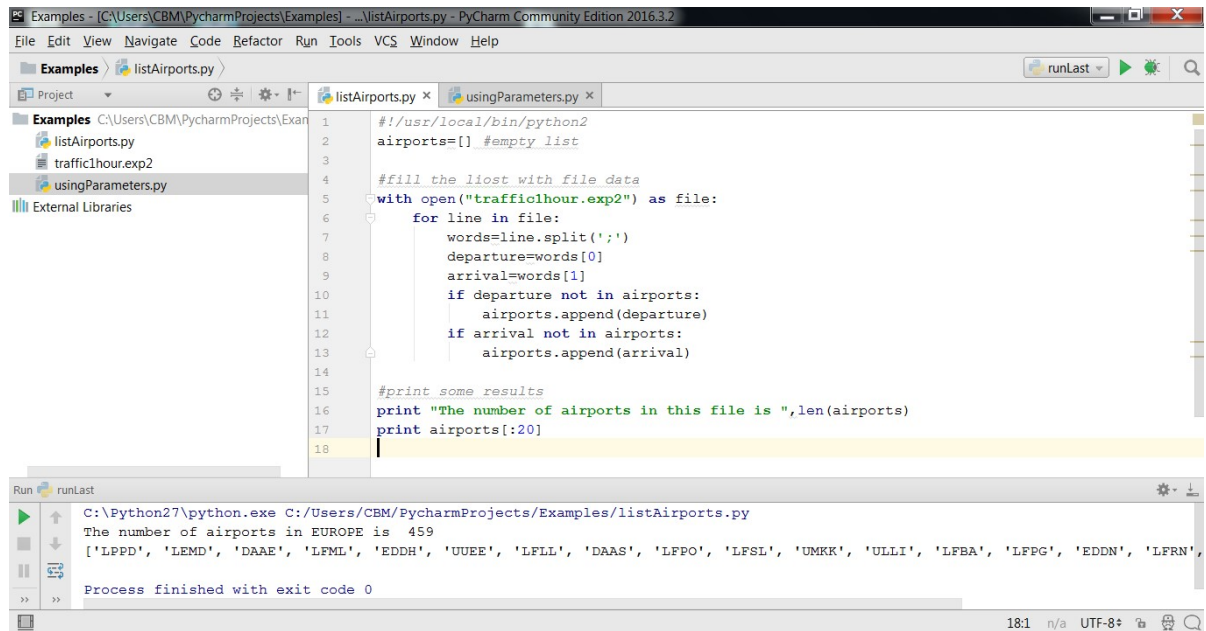
## Guide of the Activity A1 in Python

We will first work on classical algorithmic solutions. For this we will use the Python programming language available from python.org.

## Reading a local file

The code of listing 1 shows how to read a file with a regular format, line by line, and how to divide each line in words to be processed separately. We assume that the file is in the same folder than the Python program. The example program listAirports.py will read an air traffic file in tabular format and extract the list of airports contained in it.

**Listing 1:** Example of file reading and processing



The screenshot displays the PyCharm IDE interface. The main editor window shows the code for `listAirports.py`. The code reads a file named `traffic1hour.exp2` and processes it line by line, splitting each line into words and extracting departure and arrival airports. The output of the program is shown in the Run console at the bottom.

```
1  #!/usr/local/bin/python2
2  airports=[] #empty list
3
4  #fill the list with file data
5  with open("traffic1hour.exp2") as file:
6      for line in file:
7          words=line.split(';')
8          departure=words[0]
9          arrival=words[1]
10         if departure not in airports:
11             airports.append(departure)
12         if arrival not in airports:
13             airports.append(arrival)
14
15 #print some results
16 print "The number of airports in this file is ",len(airports)
17 print airports[:20]
18
```

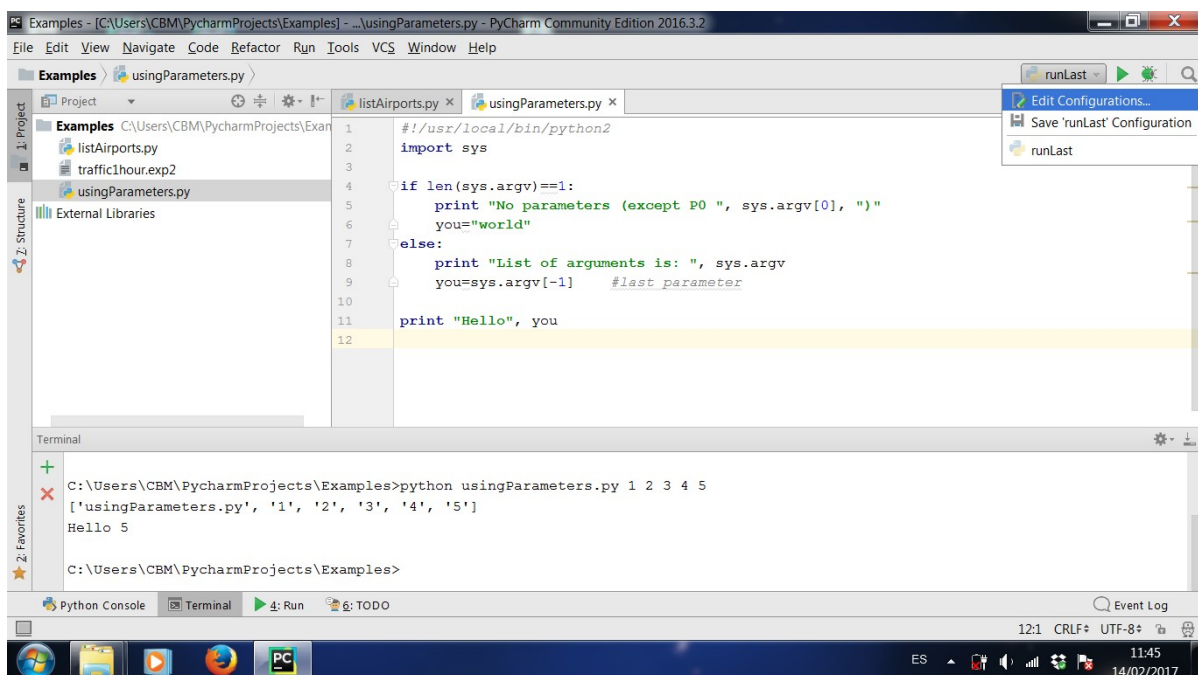
The Run console output shows the following text:

```
C:\Python27\python.exe C:/Users/CBM/PycharmProjects/Examples/listAirports.py
The number of airports in EUROPE is 459
['LPPD', 'LEMD', 'DAAE', 'LFML', 'EDDH', 'UUEE', 'LFLL', 'DAAS', 'LFPO', 'LFSL', 'UMKK', 'ULLI', 'LFBA', 'LFPG', 'EDDN', 'LFRN',
Process finished with exit code 0
```

## Passing parameters to a program

Parameters (or arguments) allow to parameterize a program by entering at run time some important value, for instance, the data file that the program shall process. To process parameters the Python module *sys* is used (it is a default library available by default). Type in a Python console the command *dir(sys)* to obtain the lists of possible functions and fields in module *sys*. In particular there is a list containing the program parameters, *argv*, which can be used as shown in Listing 2. The easiest way to enter parameters is from the terminal (as also shown in Listing 2, bottom of the windows). From an IDE, such as PyCharm, parameters can also be entered using some configuration menu linked to the run menu. In the figure, the “Edit Configuration” button (Top Right) on PyCharm.

**Listing 2:** Example of script parameters

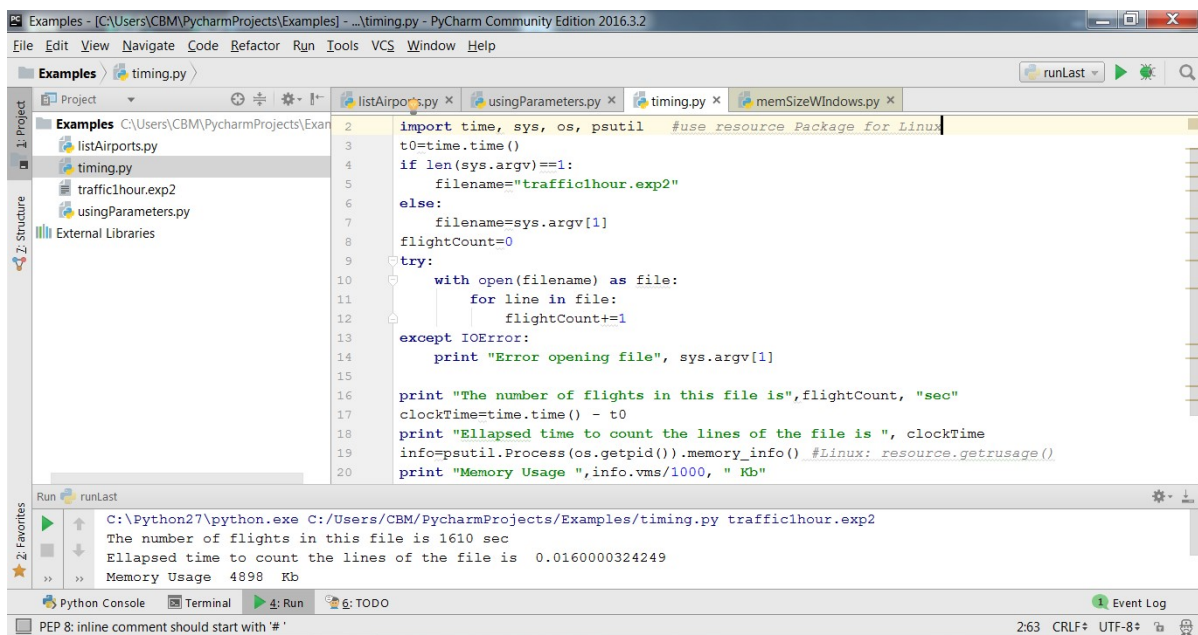


## Execution time of a program

Wall Clock Time: The Python module *time* is a default library available to measure the execution time of a program. The function *time.time()* returns the absolute clock time in seconds at the moment of invocation. Subtracting 2 successive times provides the elapsed seconds between the 2 calls.

See in Listing 3 a program that counts the lines of a file entered as parameter and show the execution time. This program also shows the code to check that the parameter used as filename exists to avoid the raise of an Exception.

**Listing 3:** Example of time and memory usage



The screenshot shows the PyCharm IDE interface. The main editor displays a Python script named `timing.py`. The script imports `time`, `sys`, `os`, and `psutil`. It checks if a command-line argument is provided; if not, it uses a default filename `traffic1hour.exp2`. It then opens the file, counts the number of lines, and prints the result. It also calculates the execution time using `time.time()` and prints it. Finally, it uses `psutil` to get memory usage information and prints it. The output window at the bottom shows the execution results for the command `C:\Python27\python.exe C:/Users/CBM/PycharmProjects/Examples/timing.py traffic1hour.exp2`. The output is: `The number of flights in this file is 1610 sec`, `Ellapsed time to count the lines of the file is 0.0160000324249`, and `Memory Usage 4898 Kb`.

```
import time, sys, os, psutil #use resource Package for Linux
t0=time.time()
if len(sys.argv)==1:
    filename="traffic1hour.exp2"
else:
    filename=sys.argv[1]
flightCount=0
try:
    with open(filename) as file:
        for line in file:
            flightCount+=1
except IOError:
    print "Error opening file", sys.argv[1]

print "The number of flights in this file is",flightCount, "sec"
clockTime=time.time() - t0
print "Ellapsed time to count the lines of the file is ", clockTime
info=psutil.Process(os.getpid()).memory_info() #Linux: resource.getrusage()
print "Memory Usage ",info.vms/1000, " Kb"
```

Run C:\Python27\python.exe C:/Users/CBM/PycharmProjects/Examples/timing.py traffic1hour.exp2  
The number of flights in this file is 1610 sec  
Ellapsed time to count the lines of the file is 0.0160000324249  
Memory Usage 4898 Kb

When locating the 2 calls to the `time()` function be careful and avoid including the input and output operations into the execution time.



## Installing new packages (Example in PyCharm)

The Python modules are available in Internet and can be download from a terminal using the *pip* command:

*pip install moduleName*

In Integrated (Program) Development Environments (IDE) such as PyCharm a module can be download without the need to open a terminal window. You shall have pip installed in any case. Check it before proceeding.

Figure 3 shows the menu options to be selected from PyCharm to download a package: first go to File->Settings, then select the option Project: Examples->Project Interpreter. Here you can select the Python version to use if you have more than one versions installed in your computer, and the installed packages. In the right there is '+' sign that you should press in order to install a new package. Just enter the name in the search box and press enter. Then select the package and push on the Install button.

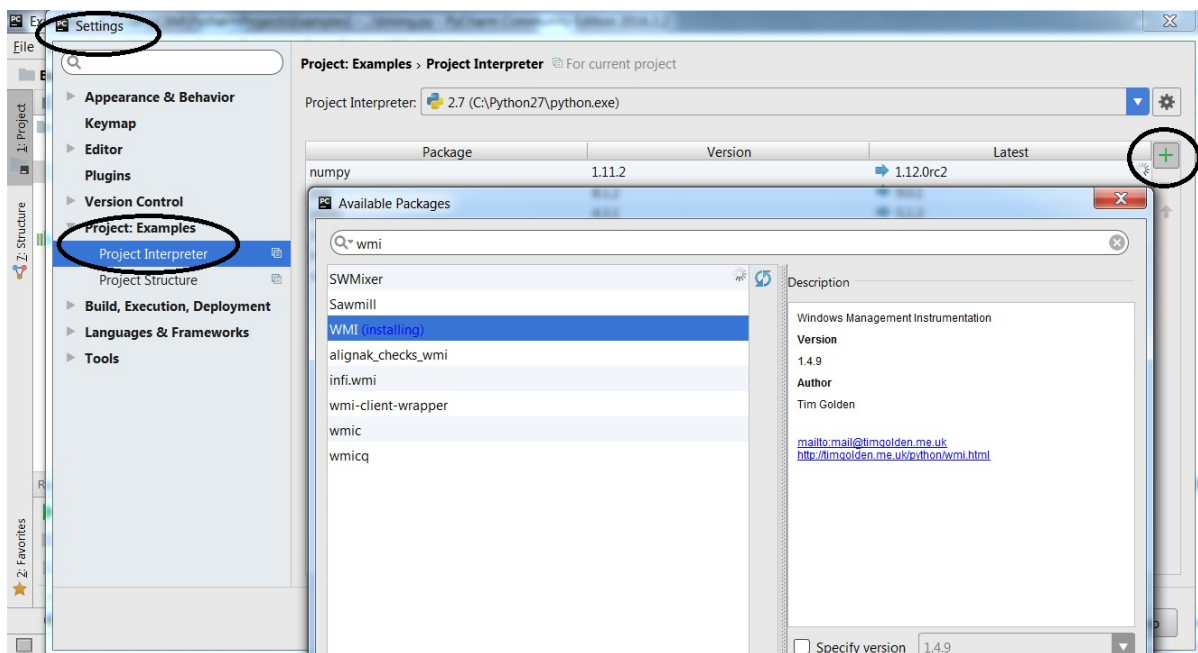


Figure 3 – Installing a package in PyCharm

---

## Accessing to Internet documents and Parsing XML

The complete data of Old Bailey is available in XML format only from the University of Sheffield's data repository (ORDA):

<http://dx.doi.org/10.15131/shef.data.4775434>

The dataset consists of 2,163 editions of the Proceedings and 475 Ordinary's Accounts marked up in TEI-XML, and contains some documentation covering the data structure and variables. Each Proceedings file represents one session of the court (1674-1913), and each Ordinary's Account file represents a single pamphlet (1676-1772). The Python modules are available in Internet and can be download from a terminal using the *pip* command:

*pip install moduleName*

In Integrated (Program) Development Environments (IDE) such as PyCharm a module can be download without the need to open a terminal window. You shall have pip installed in any case. Check it before proceeding.x

These slides show how to access the Old Bailey repositories in Python using the **request** module, the problem with the original URL and how to process the ORDA files.

---

## Guide of the Activity A1 in Pandas

This is a very brief introduction to pandas. Pandas is a library very useful to deal with tabulated data, such as a spreadsheet file. More examples can be found in this [Jupyter Notebook](#).

### Reading a file

The code below 1 shows how to read a text file in which element are organized into columns. This type of files are the CSV files (comma separated values), although other separators can be used instead of commas. This is the case of the air traffic file and we use the *sep* parameter to set the separator.

```
import pandas as pd

ap = pd.read_csv("traffic1hour.exp2", sep=';', header=None, usecols=[0,1])
ap.columns=['Dep', 'Arr']
```

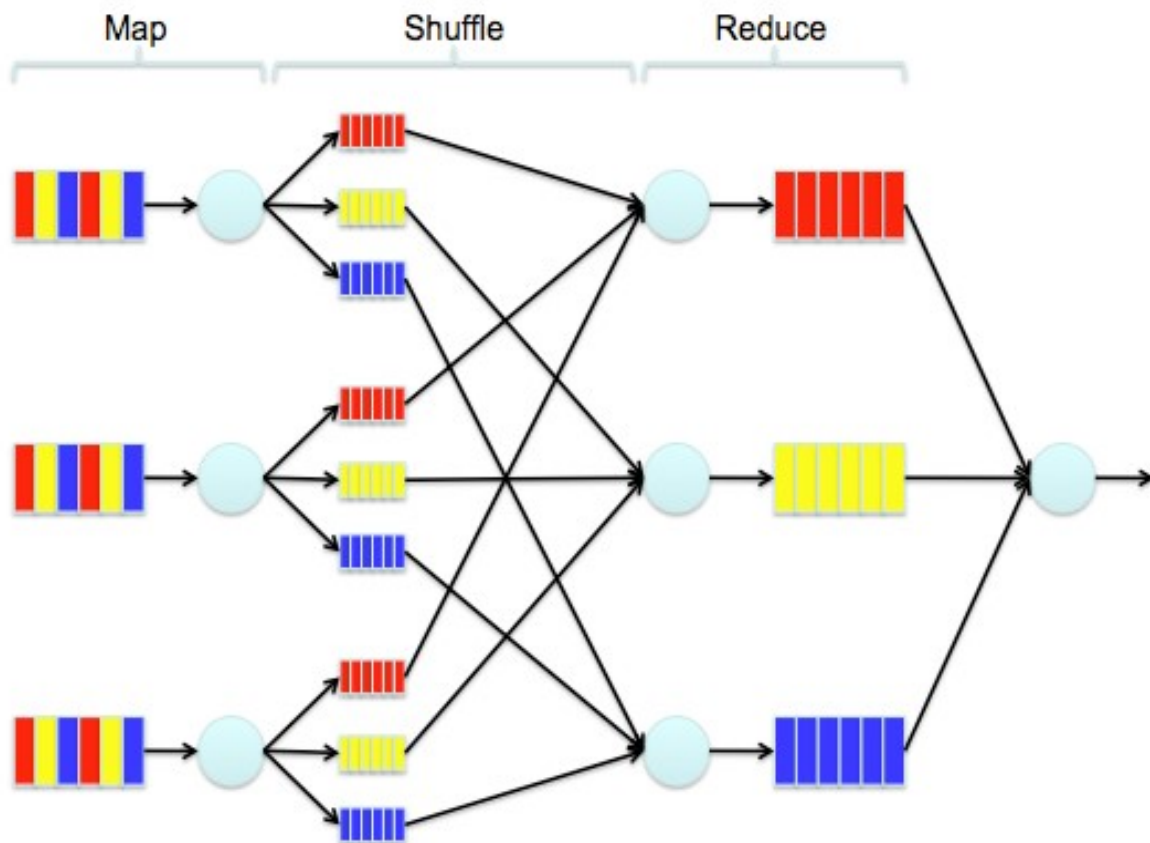
The pandas `read_csv()` function accepts also zipped CSV files and remote files through its URL.

---

## Guide of the Activity A1 in Spark

### MapReduce in Python (Sequential)

Two key concepts define Spark: the MapReduce programming model (taken from Hadoop), which can be more carefully read from the original sources from Google: paper [DeGh2010] and presentation at <http://es.slideshare.net/rantav/introduction-to-map-reduce>; and the RDD (Resilient Distributed Data) data abstraction.



Source <http://mm-tom.s3.amazonaws.com/blog/MapReduce.png>

*Figure 4: MapReduce concept*

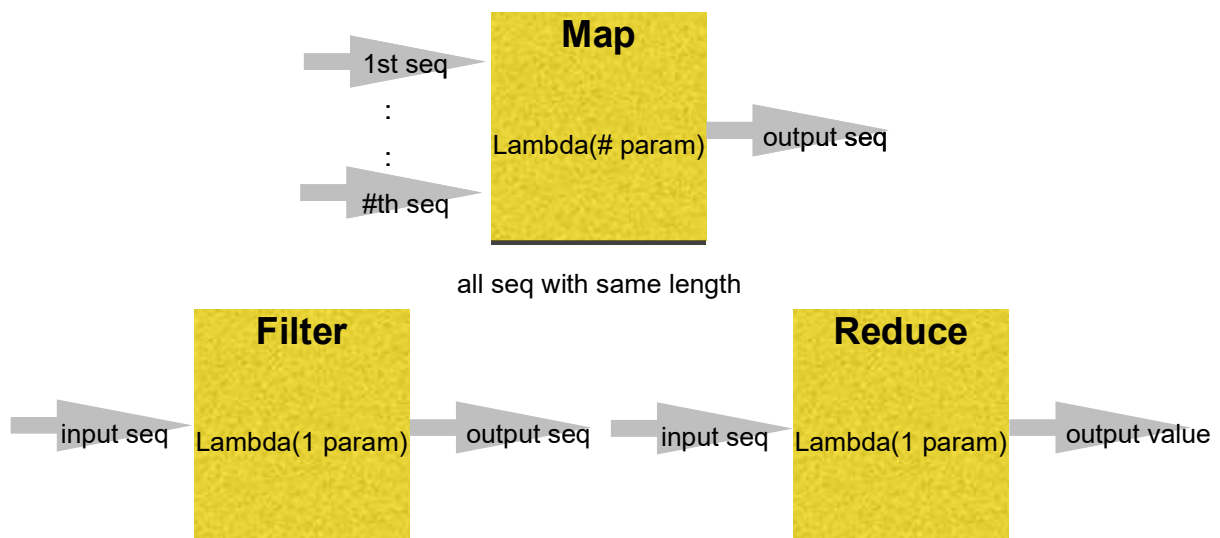
The **MapReduce** programming model is represented graphically in Figure 4. Map and Reduce must be programmed for each problem, but shuffle is managed by Spark. MapReduce uses intensively the lambda operator of Python, explained in this document at the end of the Python section. Review this section to make sure you understand it.

---

## Lambda, map, filter and reduce in Python

Python has a powerful tool, the lambda operator, very used on MapReduce algorithms, as we will see in this practice. The lambda operator is a way to express a function in a very compact way. It is especially suited for the map, filter and reduce functions. In this section we will first present these three functions using standard functions and then we will show how they combine with the lambda operator. For more information see <http://www.python-course.eu/lambda.php>

Map, filter and reduce are three functions able to process a list of elements, using a generic processing function. The schema of the three functions can be found in Figure 3.



*Figure 3: Scheme of Map, Filter and Reduce functions*

- **Map:** the map function receives 1 or more input sequences, all with the same number of elements, and produces a unique new sequence of elements as a combination of the elements, operated one-by-one, of the input sequences. The combination of the elements is given by a defined function which is also passed as input parameter to Map. This function must have the same number of input parameters than the number of input sequences of the map function. The number of elements of the output list is the same as the length of the input lists. Listing 5 shows an example of using the map function with 1, 2 and 3 input sequences.

### Listing 5: Map function usage examples

```
#some data -lists of 9 items-
list1=[1, 2, 3, 4, 5, 6, 7, 8, 9]
list2=[4, 6, 8, 2, 1, 3, 2, 7, 10]
list3=[-1, 1, -1, 1, 1, -2, 2, 1, -1]

#some auxiliary functions
def fun1 (x):
    return (x*2)

def fun2 (x,y):
    return (x+y*2)

def fun3 (x,y,z):
    return (x/z+y*2)

print ("\n#### MAP EXAMPLES")
#example of map: 1 list => 1 list (same size both). Needs a function of 1 param
list_out=map(fun1, list1)
print("multiple by 2 list1 = "+str(list_out))

#example of map: 2 lists => 1 list (same size all). Needs a function of 2 param
list_out=map(fun2, list1, list2)
print("combine list1+list2*2 = "+str(list_out))

#example of map: 3 lists => 1 list (same size all). Needs a function of 3 param
list_out=map(fun3, list1, list2, list3)
print("combine lists1-2-3 = "+str(list_out))
```

- **Filter:** the filter function receives one -and only 1- input list and produces an output list with a subset of the elements of the input. The filter function receives also an auxiliary function as parameter which specifies the filtering condition. The auxiliary function must have 1 -and only 1- input parameter, and must return as output parameter a boolean value (True/False). The auxiliary function is applied element-by-element to the input element list and if the condition given in the auxiliary function is met (returns True) then that element will be part of the output list. Otherwise the elements will be 'deleted'. Listing 6 shows an example of the use of the filter function.
- **Reduce:** the reduce function receives one -and only 1- input list and produces as output a scalar value obtained from the combination of all the elements of the input. The combination is specified in an auxiliary function, which is also a parameter of reduce. The auxiliary function is applied in a two-element-wise way to the input list. Listing 7 shows an example of the use of the reduce function.

The **lambda** operator can be used to specify the auxiliary functions in a more compact manner than using the def syntax. Listing 8 shows two ways of using the lambda operator: first using a variable to store the function, and second directly as parameter of the map/filter /reduce functions.

---

**Listing 6: Filter function example**

```
#a data -lists of 9 items-
list1=[1, 2, 3, 4, 5, 6, 7, 8, 9]

#an auxiliary function
def fun1_bool (x):
    return (x>5)

print ("\n#### FILTER EXAMPLES")
#example of filter: 1 list => 1 sub-list. Needs a boolean function of 1 param
list_out=filter(fun1_bool, list1)
print("Sub-list >5 of list1 = "+str(list_out))
```

---

**Listing 7: Reduce function example**

```
#a data -lists of 9 items-
list1=[1, 2, 3, 4, 5, 6, 7, 8, 9]

#an auxiliary function
def fun2 (x,y):
    return (x+y*2)

print ("\n#### REDUCE EXAMPLES")
#example of filter: 1 list => 1 number. Needs a function of 2 param
list_val=reduce(fun2, list1)
print("Operating consecutive elements of list1 = "+str(list_val))
```

---

**Listing 8: Using lambda operator**

```
#FIRST: alternative way to define an auxiliary function, using a variable
fun2=lambda x,y: x+y*2

#fun2 can now be used in the same way as before as map/filter/reduce parameter
list_val=reduce(fun2, list1)

#SECOND: alternative way to pass the function directly to map/filter/reduce
list_out=map(lambda x,y: x+y*2, list1, list2)
list_out2=filter(lambda x: x>5, list1)
```

---

## Data Distribution in SPARK (RDD)

In the bibliography book of this course [MaGoTo15] you have a full chapter (number 3) devoted to the **RDD** concept. RDD implements the distribution of the (big) data into a cluster of computers, obtaining at the same time efficiency and redundancy. Creating an RDD automatically distributes the data and operating RDD automatically parallelizes the processing in the nodes of the cluster, locating the code close to the data to increment efficiency. Also the data of an RDD is replicated with several levels of redundancy, making the system more fault tolerant (very important when executing in clusters of thousands of nodes, where individual failures are frequent) and giving more flexibility to the parallel code allocation.

The easy way to create an RDD for our practices is by using the `textFile` function. See listing 9 for an example of how to create an SCC from a file. For more information about RDD other than the book, consult in <http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds>

### Listing 9: Creating a RDD from a remote file, downloading it first

```
# *_ coding: utf-8 *_
import urllib.request
from pyspark import SparkContext

url = https://raw.githubusercontent.com/SparkBarcelona/libro/master/Capitulo3/quijote.txt

# Open remote file, look at its content and save locally
file_remote = urllib.request.urlopen(url)
text = file_remote.read()
print(text[:200].decode('utf-8'))
with open("Quijote.txt",w) as file_local
    file_local.write(text)

#start SPARK and read the remote file
sc = SparkContext.getOrCreate(appName="Quijote")
rdd = sc.textFile("Quijote.txt", minPartitions=4) #optional parameter minPartitions

#and now show first lines and save as HDFS (Hadoop File System)
for line in rdd.take(20):
    print (line)
rdd.saveAsTextFile("QUIJOTE_HADOOP")
```

If running in a virtual machine in CoLab, remember to install spark with

```
!pip install pyspark
```

Observe that executing Listing 9 creates a new folder QUIJOTE in which we will find as many files as the number of partitions we defined in the RDD creation (8 in this code). Each file has a part of the original document.



---

## Map Reduce in SPARK (Parallel)

Listing 10 shows the continuation of the program of listing 9. This is an example that counts the number of words of the El Quijote book using the Map, Filter and Reduce function together with the Lambda operator as in Python. The variable file is here an RDD obtained from a file and distributed all across the Workers of Spark. The operations the Map, Filter and Reduce are methods to apply to the RDD, instead of passing the input list as parameter of the functions as done in Python.

Execute line-by-line the program above using **bin/pyspark** (see last paragraph of this section) and modify it to create intermediate variables with the partial results of each function. You can see the partial results using the **take()** function as in Listing 9.

### Listing 10: Processing data with map&reduce

```
'''
Continuation of Listing 7
'''
num_words = file
    .flatMap(lambda line: line.split())
    .filter(lambda word: word != '')
    .map(lambda word: 1)
    .reduce(lambda a, b: a+b)
print ("Number of words of El Quijote is %d"%num_words)
```

The list of all the functions that can be applied to an RDD can be found in <http://spark.apache.org/docs/latest/programming-guide.html#transformations>.

In the Listing 10, two map functions are used: the first one, **flatMap()**, combines data in a unique list of data while **map()** maintains the structure of the incoming lists. To better understand the differences of the RDD transformation functions I suggest to create a short RDD, for instance using the following instruction:

```
list=[1,2, 3, 4, 5, 6, 7, 8, 9, [10, 11, 12, 13, 14, 15], 0]
rdd_data=sc.parallelize(list)
```

and then play with the map functions to see the results of each operation.

To execute spark from a terminal we can use the interactive console mode (**pyspark** program found in SPARK\_FOLDER/bin) or the batch command (**spark-submit**, also in the same folder, and accompanied by a python program file). Several modifiers are available when submitting a task to Spark, to indicate the number of nodes of a cluster that you want to use in the execution. None of these parameters need to be considered for a local execution. For executing a program from the PyCharm environment you shall have the correct definition of a number of environment variables, or you can set them at the beginning of the program as shown in Listing 11.

---

At the SPARK\_FOLDER/examples/src/main/python you can find a list of programs for test. Also at <https://github.com/SparkBarcelona/libro> you have the listing of the python examples given in the Spark book.

## REFS

[DeGh2010] J. Dean and S. Ghemawat. MapReduce: A Flexible Data Processing Tool. Communications of the ACM. Vol 53, No.1, pp.72-77. Jan 2010.

[MaGoTo15] M. Macias, M. Gomez, R. Tous, J. Torres. Introducción a Apache Spark. Editorial UOC, Nov2015.