LAB – 5

MAP REDUCING PROGRAMS USING PYTHON

MapReduce - Introduction

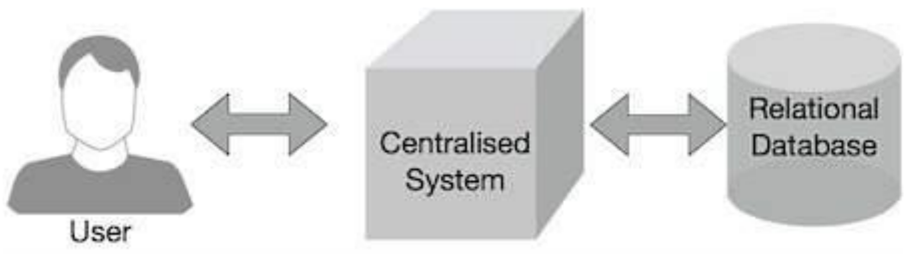
MapReduce is a programming model for writing applications that can process Big Data in parallel on multiple nodes. MapReduce provides analytical capabilities for analyzing huge volumes of complex data..

What is Big Data?

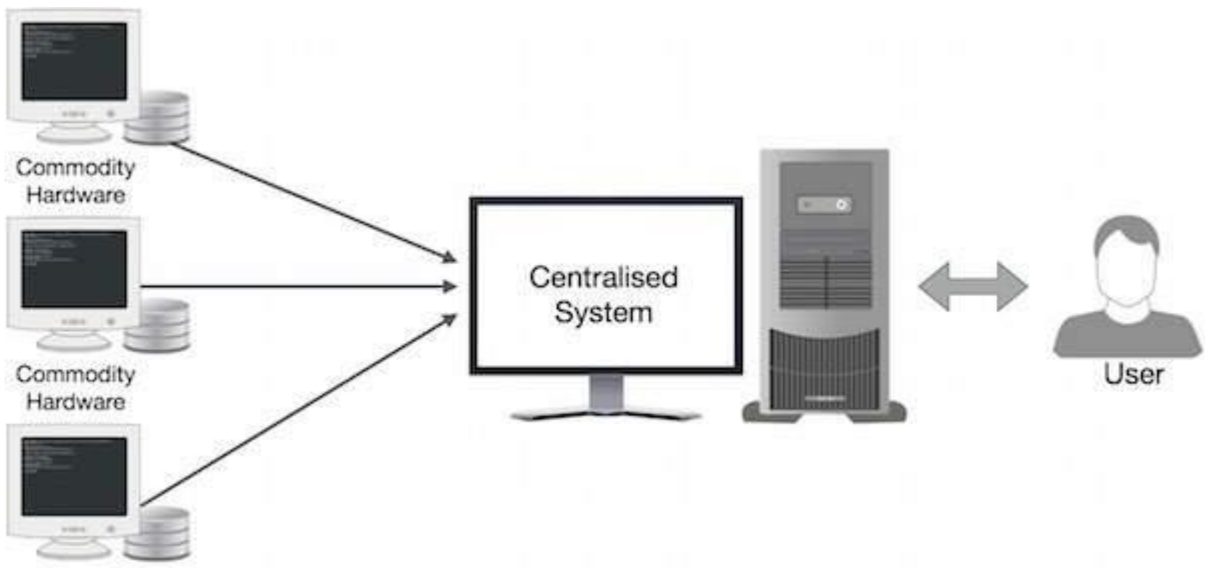
Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. For example, the volume of data Facebook or Youtube need require it to collect and manage on a daily basis, can fall under the category of Big Data. However, Big Data is not only about scale and volume, it also involves one or more of the following aspects − Velocity, Variety, Volume, and Complexity.

Why MapReduce?

Traditional Enterprise Systems normally have a centralized server to store and process data. The following illustration depicts a schematic view of a traditional enterprise system. Traditional model is certainly not suitable to process huge volumes of scalable data and cannot be accommodated by standard database servers. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.



Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.



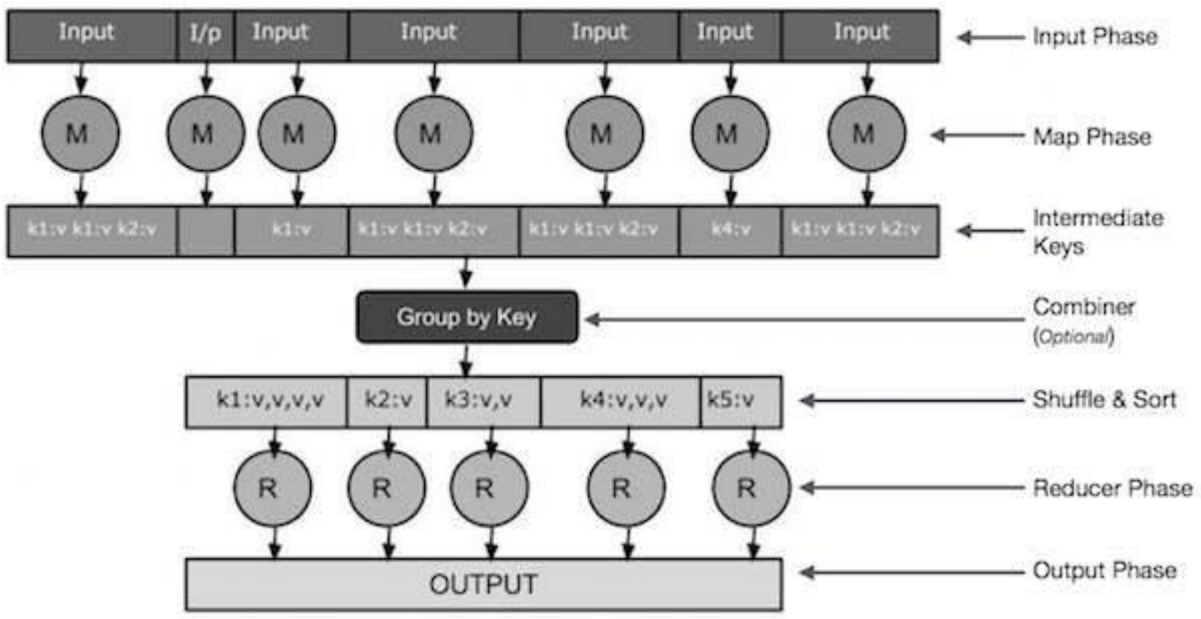
How MapReduce Works?

The MapReduce algorithm contains two important tasks, namely **Map** and **Reduce**.

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

The reduce task is always performed after the map job.

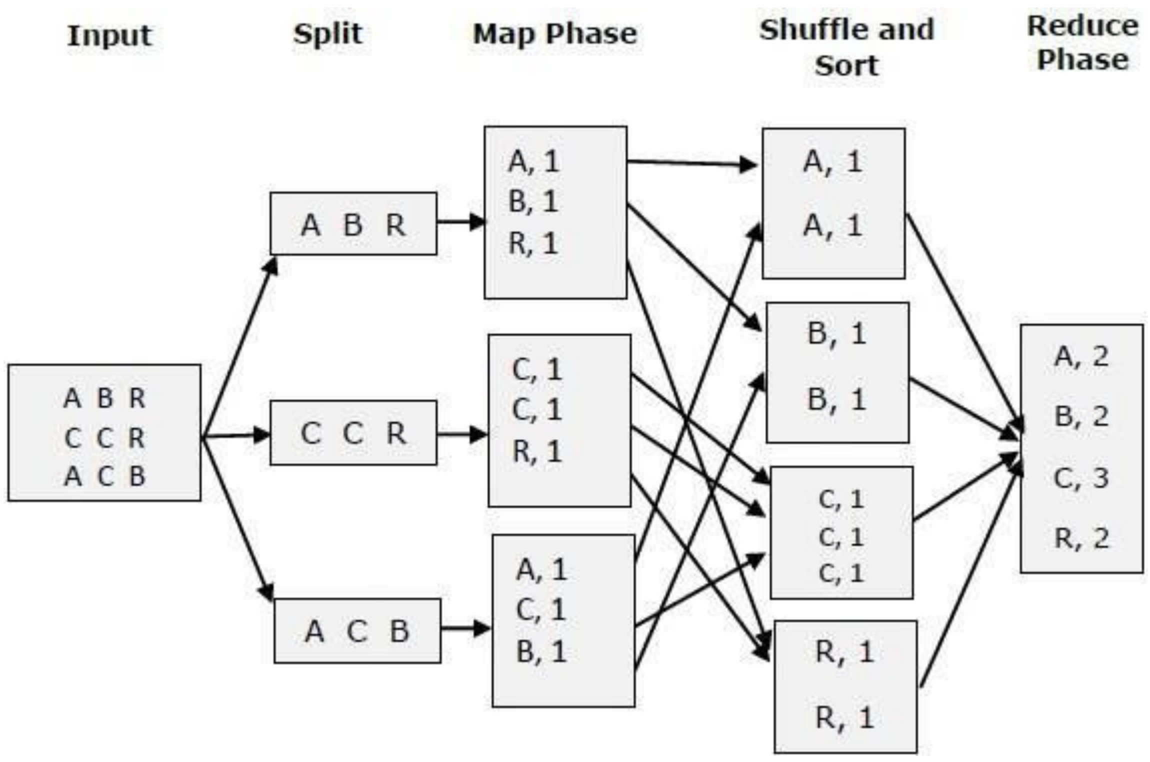
Let us now take a close look at each of the phases and try to understand their significance.



- **Input Phase** − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs
- **Map** − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** − They key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.
- **Output Phase** − In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

Let us try to understand the two tasks Map & Reduce with the help of a small diagram

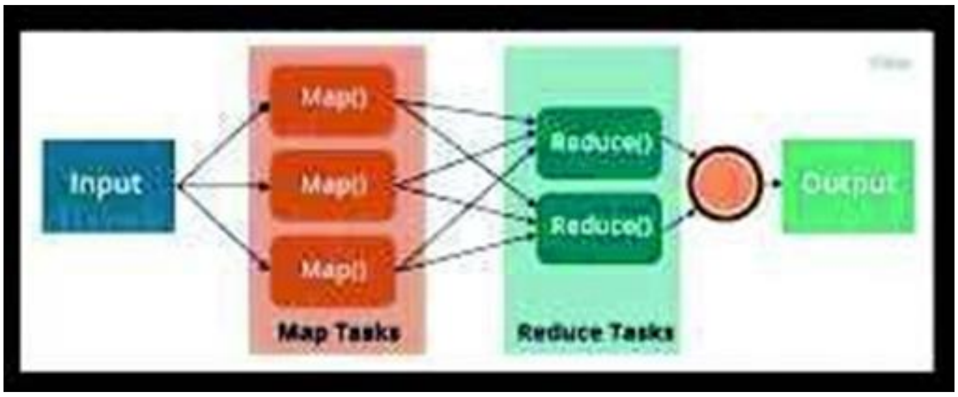


<u>MapReduce: Programming Model and Implementations</u>

- Hadoop is a framework that allows to process and store huge data sets.
- Basically, Hadoop can be divided into two parts: processing and storage.
- So, MapReduce is a programming model which allows you to process huge data stored in Hadoop.

- When you install Hadoop in a cluster, we get MapReduce as a service where you can write programs to perform computations in data in parallel and distributed fashion.

<u>Map – Reduce Implementation</u>



MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment. MapReduce consists of two distinct tasks– Map and Reduce. As the name MapReduce suggests, reducer phase takes place after mapper phase has been completed. So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs. The output of a Mapper or map job (key-value pairs) is input to the Reducer. The reducer receives the key-value pair from multiple map jobs. Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

<u>The Wordcount Example</u>

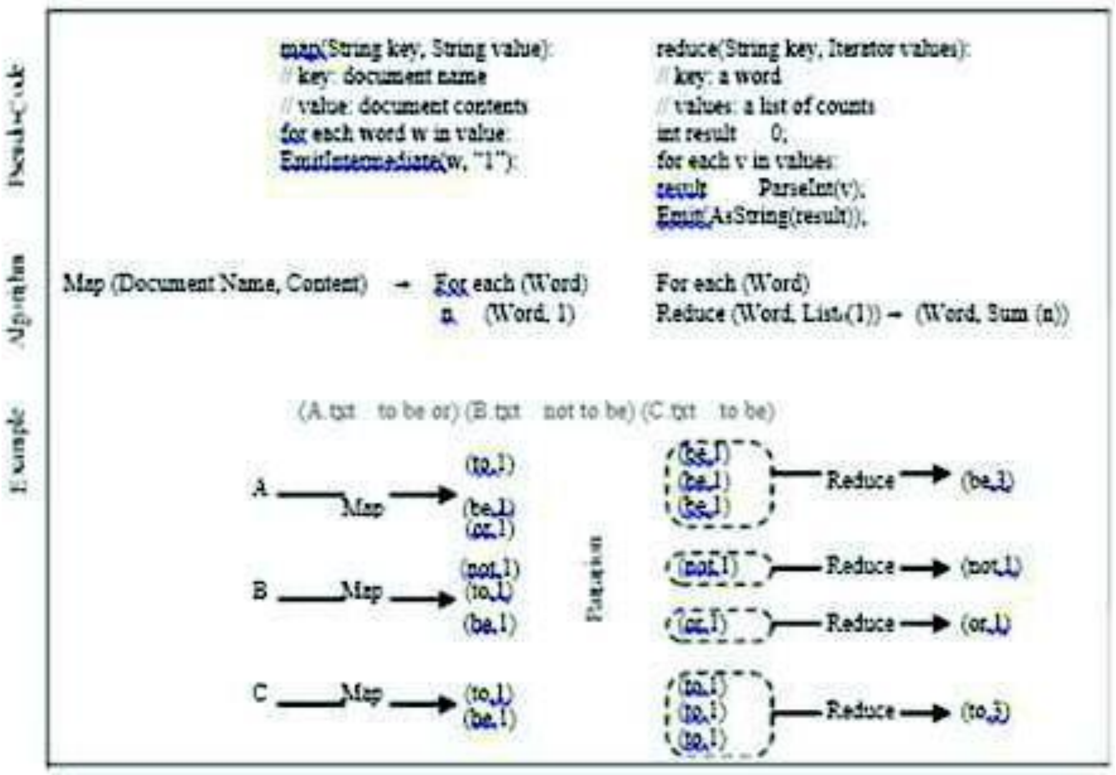
The Wordcount application counts the number of occurrences of each word in a large collection of documents.

The steps of the process are briefly described as follows:

⇨ The input is read and broken up into key/value pairs (e.g., the Map function emits a word and its associated count of occurrence, which is just "1").

⟹ The pairs are partitioned into groups for processing, and they are sorted according to their key as they arrive for reduction.

⟹ Finally, the key/value pairs are reduced, once for each unique key in the sorted list, to produce a combined result (e.g., the Reduce function sums all the counts emitted for a particular word).



```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result    0;
for each v in values:
result      ParseInt(v);
Emit(AsString(result));
```

```
Map (Document Name, Content)  →  For each (Word)
                                 n   (Word, 1)
```

```
For each (Word)
Reduce (Word, List(1)) → (Word, Sum (n))
```

Another Example

```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");
```

<“Sam”, “1”>, <“Apple”, “1”>, <“Sam”, “1”>,

<“Mom”, “1”>, <“Sam”, “1”>, <“Mom”, “1”>,

```
reduce(String output_key, Iterator intermediate_values):
  // output_key: a wo
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
```

<“Sam” , [“1”,”1”,”1”]>, <“Apple” , [“1”]>,

<“Mom” , [“1”, “1”] >

“3”

“1”

“2”

1. Write a basic wordcount program.

Sample Pseudocode:

Mapper:

```
void Map (key, value)
{
        for each word x in value:
              emit(x, 1);
}
```

Reducer:

```
void Reduce (keyword, <list_val>)
{
   for each x in <list_val>:
        sum+=x;
        emit(keyword, sum);
}
```

PYTHON PROGRAMS

# mapper.py

```
import sys
# input comes from STDIN (standard input)

for line in sys.stdin:

    # remove leading and trailing whitespace

    line = line.strip()

    # split the line into words

    words = line.split()
    # increase counters

    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py #
        # tab-delimited; the trivial word count is 1

        print("%s\t\t%s" %(word, 1))
```

# Reducer.py

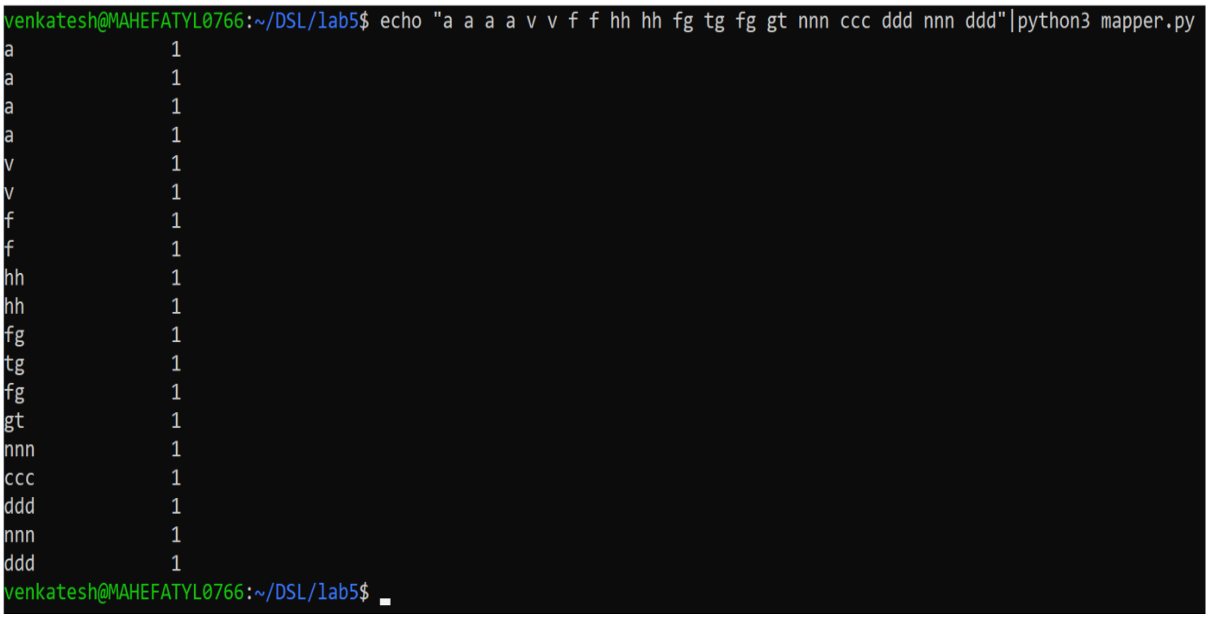
```
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN

for line in sys.stdin:

    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer

    if current_word == word:
        current_count += count
    else:

        if current_word:

            # write result to STDOUT
            print('%s\t%s' % (current_word, current_count) )

        current_count = count
        current_word = word

    # do not forget to output the last word if needed!

    if current_word == word:

        print('%s\t%s' % (current_word, current_count))
```

Test your code locally

# Test mapper.py and reducer.py locally first

1) *# very basic test (using only mapper.py)*

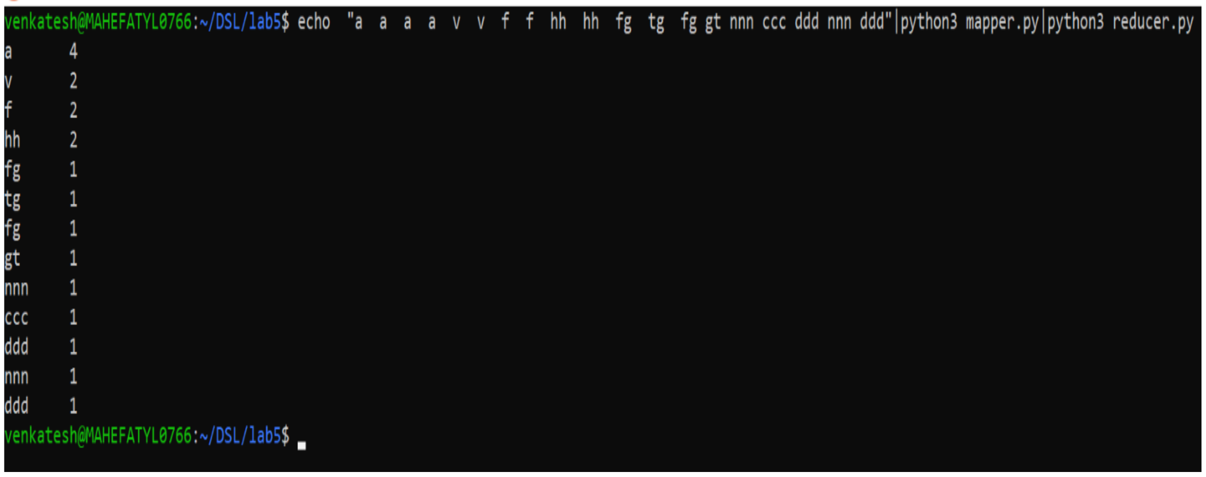
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "a a a a v v f f hh hh fg tg fg gt nnn ccc ddd nnn ddd"|python3 mapper.py



```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "a a a a v v f f hh hh fg tg fg gt nnn ccc ddd nnn ddd"|python3 mapper.py
a               1
a               1
a               1
a               1
v               1
v               1
f               1
f               1
hh              1
hh              1
fg              1
tg              1
fg              1
gt              1
nnn             1
ccc             1
ddd             1
nnn             1
ddd             1
venkatesh@MAHEFATYL0766:~/DSL/lab5$
```

2) *# very basic test (using mapper.py and reducer.py)*

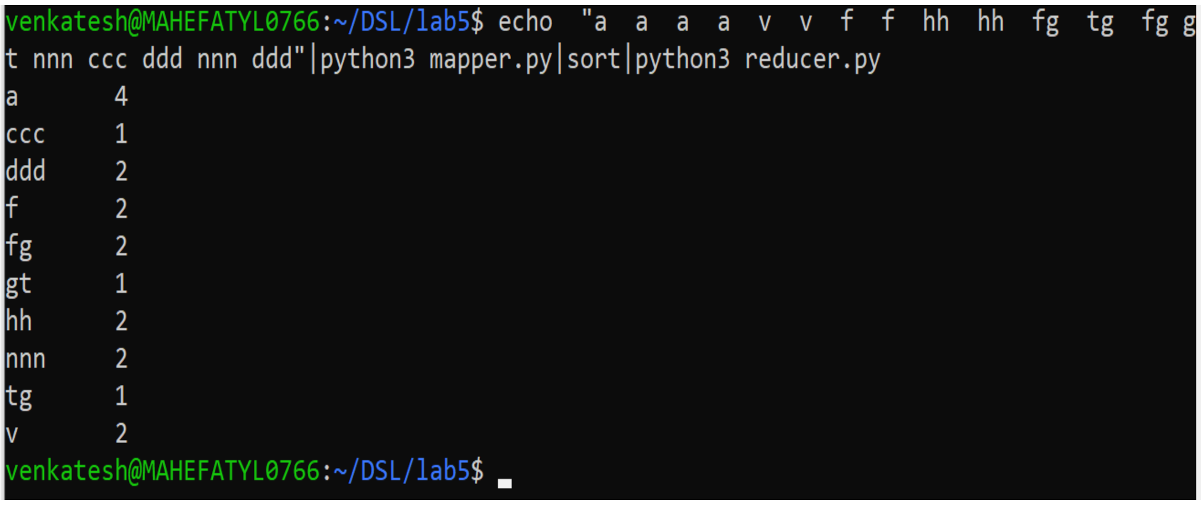
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "a a a a v v f f hh hh fg tg fg gt nnn ccc ddd nnn ddd"|python3 mapper.py|python3 reducer.py



```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo  "a  a  a  a  v  v  f  f  hh  hh  fg  tg  fg gt nnn ccc ddd nnn ddd"|python3 mapper.py|python3 reducer.py
a       4
v       2
f       2
hh      2
fg      1
tg      1
fg      1
gt      1
nnn     1
ccc     1
ddd     1
nnn     1
ddd     1
venkatesh@MAHEFATYL0766:~/DSL/lab5$
```

3) *# very basic test (use mapper.py , sort the output and use reducer.py)*

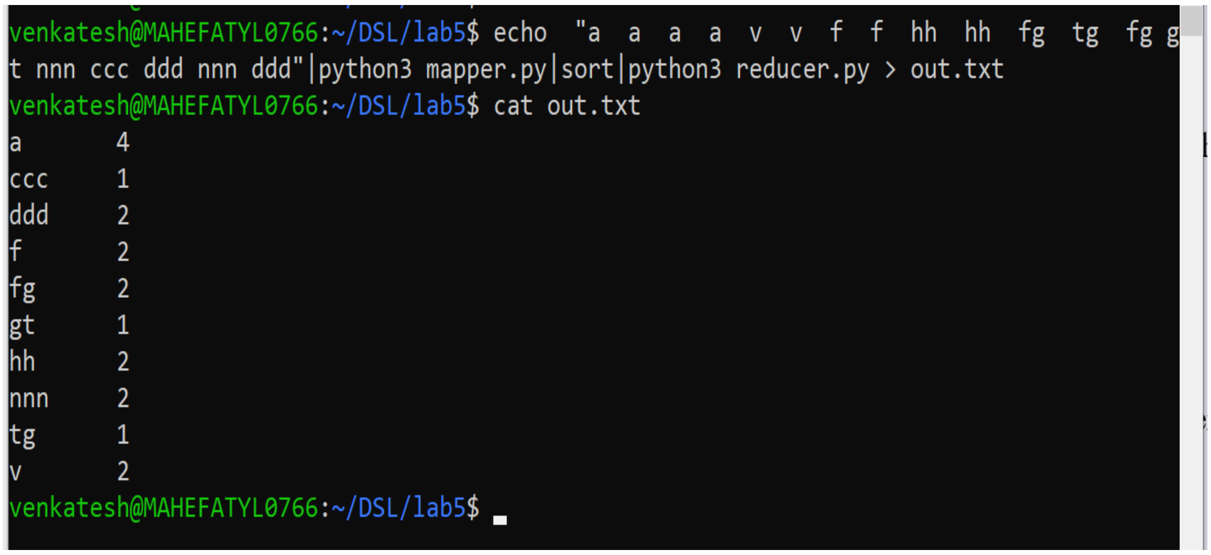
hduser@ubuntu:~$ echo "a a a a v v f f hh hh fg tg fg gt nnn ccc ddd nnn ddd"|python3 mapper.py|sort|python3 reducer.py



```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo  "a  a  a  a  v  v  f  f  hh  hh  fg  tg  fg g
t nnn ccc ddd nnn ddd"|python3 mapper.py|sort|python3 reducer.py
a       4
ccc     1
ddd     2
f       2
fg      2
gt      1
hh      2
nnn     2
tg      1
v       2
venkatesh@MAHEFATYL0766:~/DSL/lab5$
```

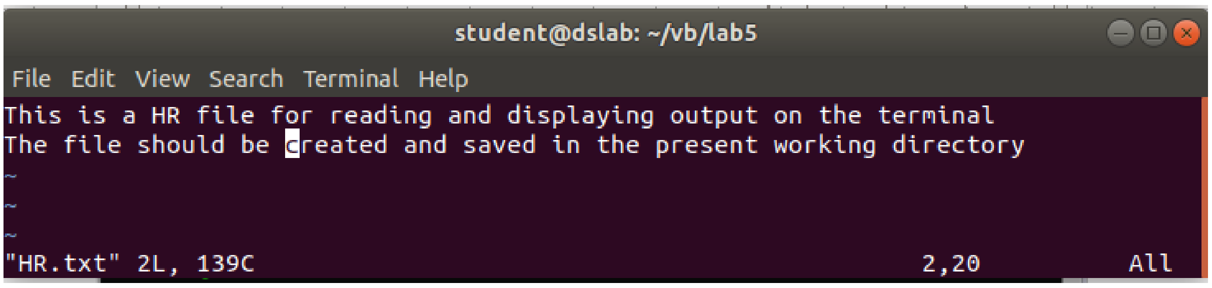
4) # very basic test (use mapper.py , sort the output and use reducer.py) and write it to text file)

venkatesh@MAHEFATYL0766*:~/DSL/lab5$ echo "a a a a v v f f hh hh fg tg fg gt nnn ccc ddd nnn ddd"|python3 mapper.py|sort|python3 reducer.py > out.txt*


```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo  "a  a  a  a  v  v  f  f  hh  hh  fg  tg  fg g
t nnn ccc ddd nnn ddd"|python3 mapper.py|sort|python3 reducer.py > out.txt
venkatesh@MAHEFATYL0766:~/DSL/lab5$ cat out.txt
a       4
ccc     1
ddd     2
f       2
fg      2
gt      1
hh      2
nnn     2
tg      1
v       2
venkatesh@MAHEFATYL0766:~/DSL/lab5$ 
```

5) # To extract words from any dataset or any file…. (use the proper path of file in the command)

Create a sample Hr.txt file as follows



```
This is a HR file for reading and displaying output on the terminal
The file should be created and saved in the present working directory
~
~
~
"HR.txt" 2L, 139C                                         2,20          All
```

Run the below command:

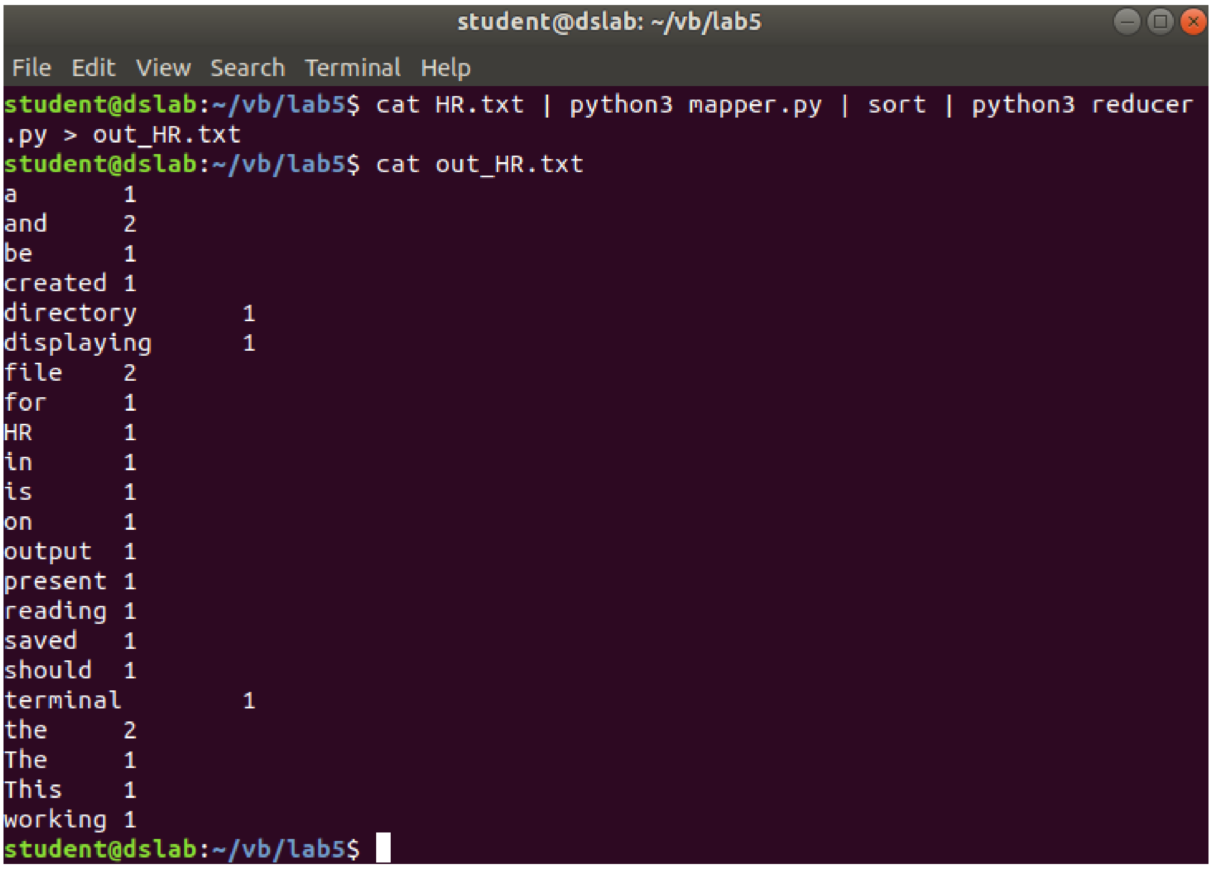
shduser@ubuntu:~$ cat /home/xxx/Desktop/HR.txt | python3 mapper.py | sort | python3

reducer.py > out_HR.txt

OR
cat HR.txt | python3 mapper.py | sort | python3 reducer.py > out_HR.txt

shduser@ubuntu:~$ cat out_HR.txt

SAMPLE OUTPUT



```
student@dslab:~/vb/lab5$ cat HR.txt | python3 mapper.py | sort | python3 reducer
.py > out_HR.txt
student@dslab:~/vb/lab5$ cat out_HR.txt
a       1
and     2
be      1
created 1
directory       1
displaying      1
file    2
for     1
HR      1
in      1
is      1
on      1
output  1
present 1
reading 1
saved   1
should  1
terminal        1
the     2
The     1
This    1
working 1
student@dslab:~/vb/lab5$ 
```

2. MapReduce program to find frequent words

# freqmap1.py

```
# MapReduce program to find frequent words
#!/usr/bin/env python
# A basic mapper function/program that
# takes whatever is passed on the input and
# outputs tuples of all the words formatted
# as (word, 1)

from __future__ import print_function
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:

    # create tuples of all words in line
    L = [ (word.strip().lower(), 1 ) for word in line.strip().split() ]

    # increase counters
    for word, n in L:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print( '%s\t%d' % (word, n) )
```

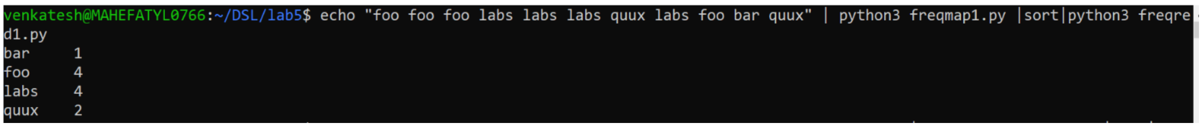
# freqred1.py

```
#!/usr/bin/env python
# reducer.py

from __future__ import print_function
import sys

lastWord = None
sum = 0

for line in sys.stdin:
    word, count = line.strip().split('\t', 1)
    count = int(count)

    if lastWord==None:
        lastWord = word
        sum = count
        continue

    if word==lastWord:
        sum += count
    else:
        print( "%s\t%d" % ( lastWord, sum ) )
        sum = count
        lastWord = word

        # output last word

if lastWord == word:
    print( '%s\t%s' % (lastWord, sum ) )
```

Sample execution

venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqred1.py

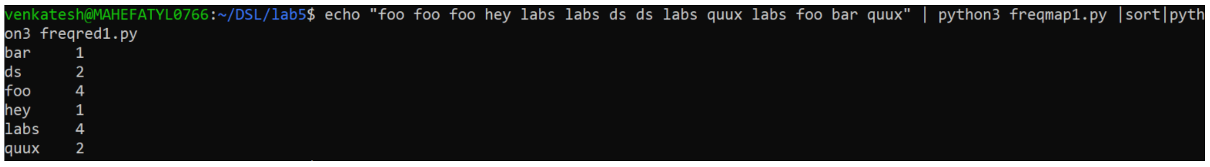
Output-1:



```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqre
d1.py
bar     1
foo     4
labs    4
quux    2
```

venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo hey labs labs ds ds labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqred1.py

Output-2:



```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo hey labs labs ds ds labs quux labs foo bar quux" | python3 freqmap1.py |sort|pyth
on3 freqred1.py
bar     1
ds      2
foo     4
hey     1
labs    4
quux    2
```

# freqmap2.py

```
#!/usr/bin/env python
# A basic mapper function/program that

# takes whatever is passed on the input and
# outputs tuples of all the words formatted
# as (word, 1)

from __future__ import print_function
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:

    word, count = line.strip().split('\t', 1)
    count = int(count)
    print( '%d\t%s' % (count, word) )
```

# freqred2.py

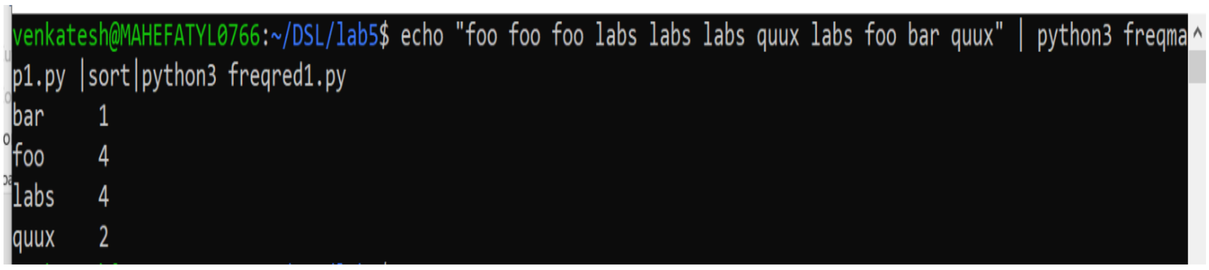
```
#!/usr/bin/env python
# reducer.py
from __future__ import print_function
import sys

mostFreq = []
currentMax = -1

for line in sys.stdin:
    count, word = line.strip().split('\t', 1)
    count = int(count)
    if count > currentMax:
        currentMax = count
        mostFreq = [ word ]
    elif count == currentMax:
        mostFreq.append( word )

# output mostFreq word(s)
for word in mostFreq:
    print( '%s\t%s' % ( word, currentMax ) )
```

Sample executions

Command-1:

```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqred1.py
```

Output-1:

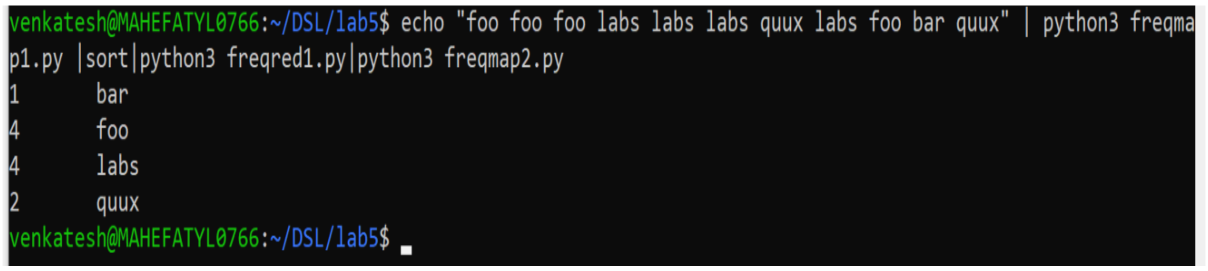


```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqma
p1.py |sort|python3 freqred1.py
bar     1
foo     4
labs    4
quux    2
```

Command-2:

venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqred1.py | python3 freqmap2.py

Output-2:

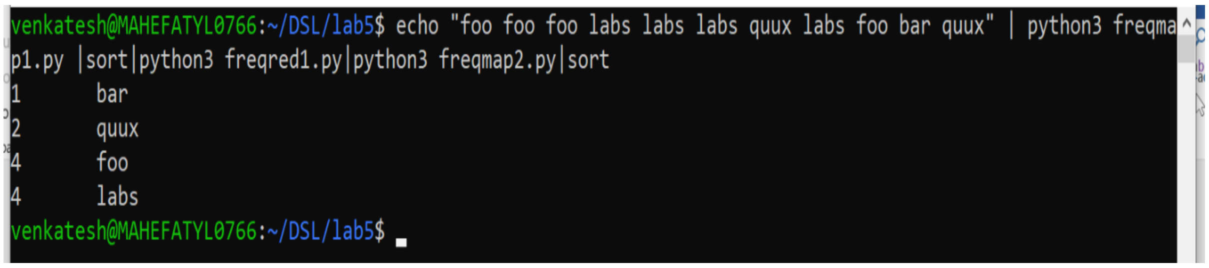


```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqma
p1.py |sort|python3 freqred1.py|python3 freqmap2.py
1       bar
4       foo
4       labs
2       quux
venkatesh@MAHEFATYL0766:~/DSL/lab5$
```

Command-3:

venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqred1.py|python3 freqmap2.py|sort

Output-3:

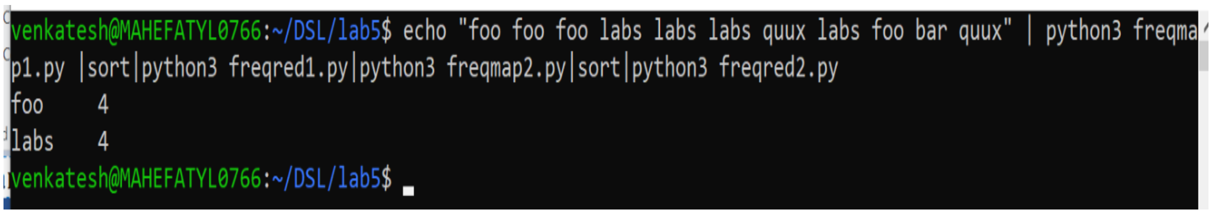


```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqma
p1.py |sort|python3 freqred1.py|python3 freqmap2.py|sort
1       bar
2       quux
4       foo
4       labs
venkatesh@MAHEFATYL0766:~/DSL/lab5$
```

Command-4:

venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqmap1.py |sort|python3 freqred1.py|python3 freqmap2.py|sort|python3 freqred2.py

Output-4:



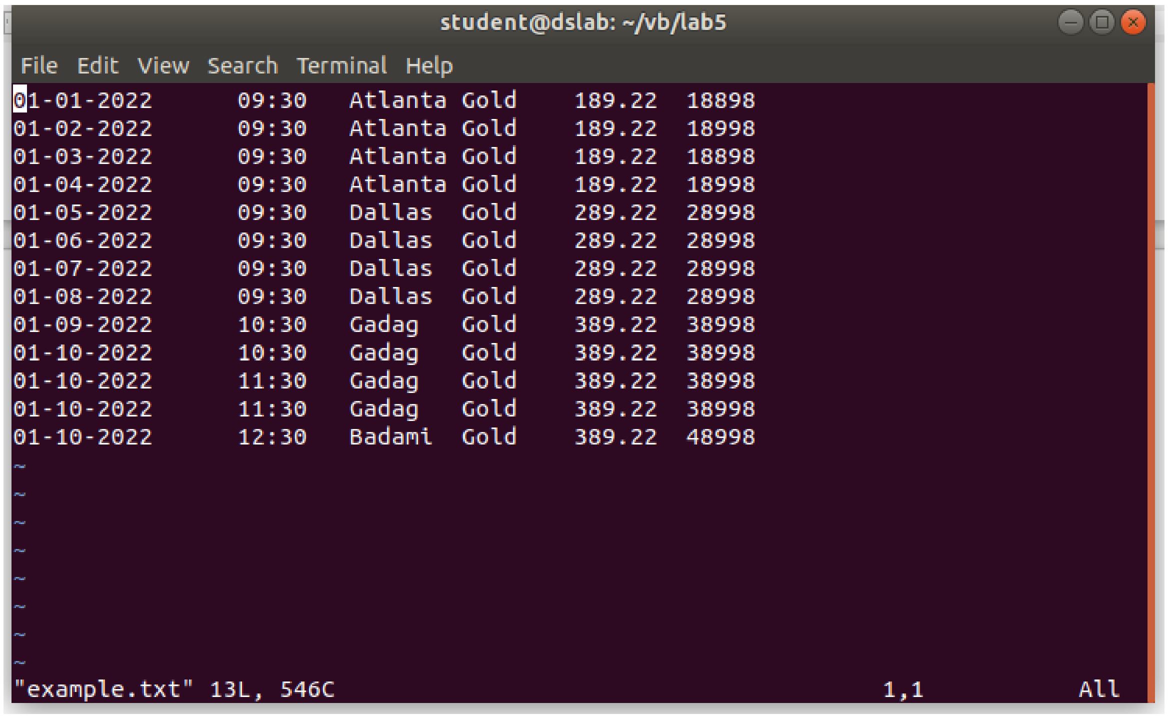
```
venkatesh@MAHEFATYL0766:~/DSL/lab5$ echo "foo foo foo labs labs labs quux labs foo bar quux" | python3 freqma
p1.py |sort|python3 freqred1.py|python3 freqmap2.py|sort|python3 freqred2.py
foo     4
labs    4
venkatesh@MAHEFATYL0766:~/DSL/lab5$
```

3. MapReduce program to explore the dataset and perform the filtering (typically creating key/value pairs) by mapper and perform the count and summary operation on the instances.

Itemmap.py

```
#import string
import fileinput
for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) == 6:
        date, time, location, item, cost, payment = data
        print ("{0}\t{1}".format(location, cost))
```

Create a example.txt file with the following data as shown below.

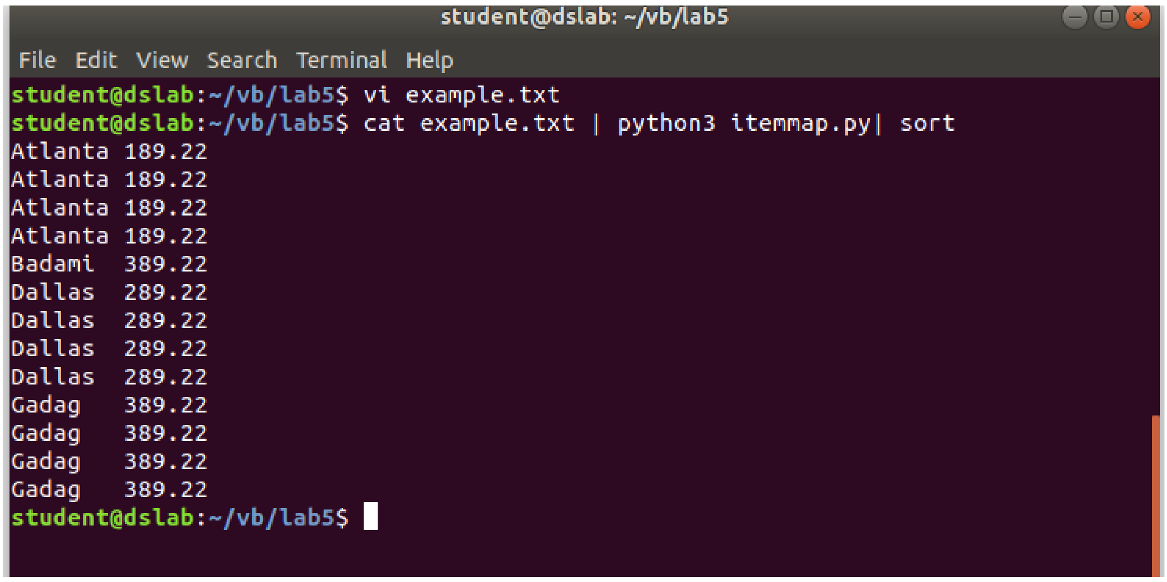


```
01-01-2022      09:30   Atlanta Gold    189.22  18898
01-02-2022      09:30   Atlanta Gold    189.22  18998
01-03-2022      09:30   Atlanta Gold    189.22  18898
01-04-2022      09:30   Atlanta Gold    189.22  18998
01-05-2022      09:30   Dallas  Gold    289.22  28998
01-06-2022      09:30   Dallas  Gold    289.22  28998
01-07-2022      09:30   Dallas  Gold    289.22  28998
01-08-2022      09:30   Dallas  Gold    289.22  28998
01-09-2022      10:30   Gadag   Gold    389.22  38998
01-10-2022      10:30   Gadag   Gold    389.22  38998
01-10-2022      11:30   Gadag   Gold    389.22  38998
01-10-2022      11:30   Gadag   Gold    389.22  38998
01-10-2022      12:30   Badami  Gold    389.22  48998
~
~
~
~
~
~
~
~
"example.txt" 13L, 546C                                       1,1           All
```

Mapper output

student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap.py| sort

OUTPUT



```
student@dslab:~/vb/lab5$ vi example.txt
student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap.py| sort
Atlanta 189.22
Atlanta 189.22
Atlanta 189.22
Atlanta 189.22
Badami  389.22
Dallas  289.22
Dallas  289.22
Dallas  289.22
Dallas  289.22
Gadag   389.22
Gadag   389.22
Gadag   389.22
Gadag   389.22
student@dslab:~/vb/lab5$
```

Itemred.py

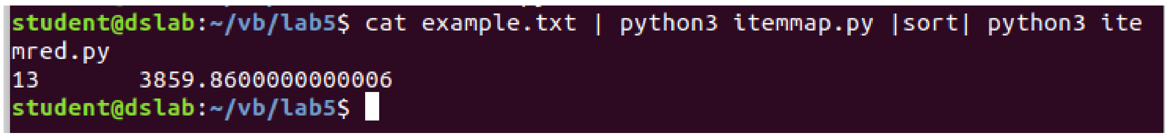
```
import fileinput
transactions_count = 0
sales_total = 0

for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) != 2:
    # Something has gone wrong. Skip this line.
        continue

    current_key, current_value  = data
    transactions_count +=  1
    sales_total  += float(current_value)
print (transactions_count, "\t", sales_total)
```

Command

student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap.py |sort| python3 itemred.py

OUTPUT

Note: Total number of instances and its sum is displayed



```
student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap.py |sort| python3 ite
mred.py
13      3859.8600000000006
student@dslab:~/vb/lab5$
```

4. Write a mapper and reducer program for word count by defining separator instead of using “\t”.

sepmap.py

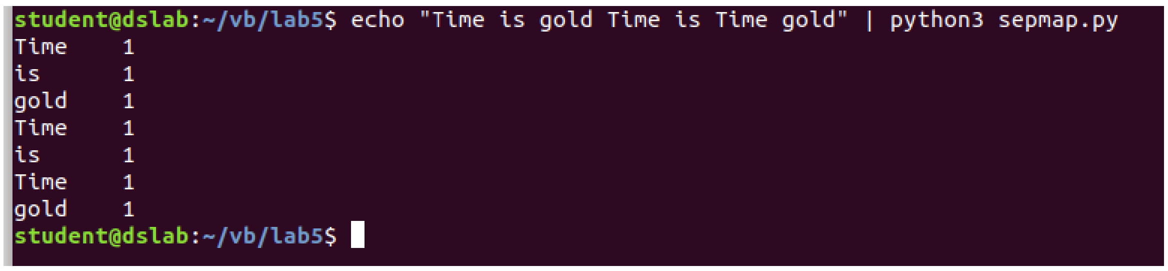
```
# sepmap.py
# A more advanced Mapper, using Python iterators and generators
import sys
def read_input(file):
    for line in file:
        # split the line into words
        yield line.split()

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        # tab-delimited; the trivial word count is 1
        for word in words:
            print('%s%s%d' % (word, separator, 1))

if __name__=="__main__":
    main()
```

```
# sepred.py
# more advanced Reducer, using Python iterators and generators
from itertools import groupby
from operator import itemgetter
import sys
def read_mapper_output(file, separator='\t'):
  for line in file:
    yield line.rstrip().split(separator, 1)


def main(separator='\t'):
   # input comes from STDIN (standard input)
   data = read_mapper_output(sys.stdin, separator=separator)
   # groupby groups multiple word-count pairs by word,
   # and creates an iterator that returns consecutive keys and their group:
   # current_word - string containing a word (the key)
   # group - iterator yielding all ["&lt;current_word&gt;", "&lt;count&gt;"]
items
   for current_word, group in groupby(data, itemgetter(0)):
      try:
         total_count = sum(int(count) for current_word, count in group)
         print ("%s%s%d" % (current_word, separator, total_count))


      except ValueError:
         # count was not a number, so silently discard this item
         pass
if __name__=="__main__":
   main()
```

Command -1

student@dslab:~/vb/lab5$ echo "Time is gold Time is Time gold" | python3 sepmap.py

OUTPUT-1

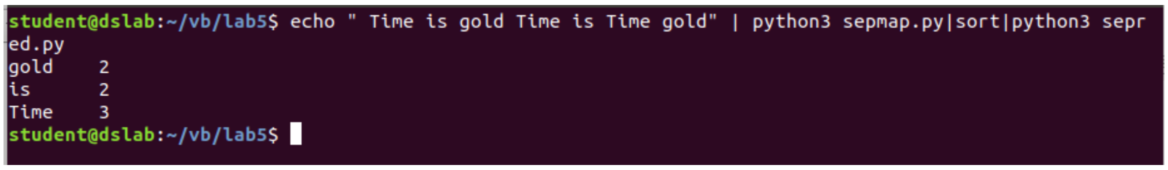


```
student@dslab:~/vb/lab5$ echo "Time is gold Time is Time gold" | python3 sepmap.py
Time	1
is	1
gold	1
Time	1
is	1
Time	1
gold	1
student@dslab:~/vb/lab5$ 
```

Command -2

student@dslab:~/vb/lab5$ echo " Time is gold Time is Time gold" | python3 sepmap.py|sort|python3 sepred.py

OUTPUT-2



```
student@dslab:~/vb/lab5$ echo " Time is gold Time is Time gold" | python3 sepmap.py|sort|python3 sepr
ed.py
gold	2
is	2
Time	3
student@dslab:~/vb/lab5$ 
```

Execution of the above on a sample file (Exercise problem)

```
student@dslab:~/vb/lab5$ cat example-vb.txt | python3 sepmap.py |python3 sepred.py
```

Obtained Sample output

```
student@dslab:~/vb/lab5$ cat example-vb.txt | python3 sepmap.py |python3 sepred.py
01-01-2022	1
09:30	1
Atlanta	1
Gold	1
189.221
18898	1
01-02-2022	1
09:30	1
Atlanta	1
Gold	1
189.221
18998	1
01-03-2022	1
09:30	1
Atlanta	1
Gold	1
189.221
18898	1
01-04-2022	1
09:30	1
Atlanta	1
Gold	1
```

189.221

18998 1

01-05-2022 1

09:30 1

Dallas 1

Gold 1

289.221

28998 1

01-06-2022 1

09:30 1

Dallas 1

Gold 1

289.221

28998 1

01-07-2022 1

09:30 1

Dallas 1

Gold 1

289.221

28998 1

01-08-2022 1

09:30 1

Dallas 1

Gold 1

289.221

28998 1

01-09-2022 1

10:30 1

Gadag 1

```
Gold	1
389.221
38998	1
01-10-2022	1
10:30	1
Gadag	1
Gold	1
389.221
38998	1
01-10-2022	1
11:30	1
Gadag	1
Gold	1
389.221
38998	1
01-10-2022	1
11:30	1
Gadag	1
Gold	1
389.221
38998	1
01-10-2022	1
12:30	1
Badami	1
Gold	1
389.221
48998	1
student@dslab:~/vb/lab5$
```

5. Write a map reduce program that returns the cost of the item that is most expensive, for each location in the dataset example.txt

itemmap_expensive.py

```
import fileinput

for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data)==6:
      date, time, location, item, cost, payment = data
      print("{0}\t{1}".format(location, cost))
```

Note: Input for this program is ***example.txt*** file.

itemred_expensive.py

```
import fileinput

max_value = 0
old_key = None

for line in fileinput.input():
    data = line.strip().split("\t")
    if len(data) != 2:
        # Something has gone wrong. Skip this line.
        continue

    current_key, current_value = data
    # print(current_key, current_value)

    # Refresh for new keys (i.e. locations in the example context)
    if old_key and old_key != current_key:
      print(old_key,"\t", max_value)
      max_value = 0

    if float(current_value) > float(max_value):
      max_value = float(current_value)
    old_key = current_key

if old_key != None:
  print (old_key, "\t", max_value)
```

COMMAND-1

```
student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap_expensive.py
```

OUTPUT-1

```
Las Vegas	208.97
Miami	84.11
Tucson	489.93
San Francisco	388.3
Dallas	145.63
Tampa	353.23
Washington	481.31
San Jose	492.8
Newark	410.37
Memphis	354.44
Jersey City	369.07
Plano	4.65
Buffalo	337.35
Louisville	213.64
Miami	154.64
Los	164.5
Birmingham	1.64
Mesa	13.79
Wichita	158.25
Indianapolis	152.77
San Bernardino	332.43
Indianapolis	464.36
Stockton	180.61
Austin	48.09
Buffalo	386.56
Santa Ana	2.75
Gilbert	11.31
```

```
New York	221.35
Corpus Christi	157.91
Riverside	349.41
Chicago	364.53
Fremont	404.17
Rochester	460.39
Raleigh	61.22
Chicago	431.73
Cincinnati	288.32
Rochester	342.62
Pittsburgh	498.29
Rochester	485.71
Glendale	14.09
Cincinnati	1.41
Irvine	15.19
Boston	397.21
Scottsdale	214.32
Atlanta	189.22
Cincinnati	443.78
Lubbock	27.68
Cincinnati	129.6
Santa Ana	282.13
Aurora	82.38
student@dslab:~/vb/lab5$
```

COMMAND-2

```
student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap_expensive.py |
sort
```

OUTPUT-2

```
Atlanta	189.22
Aurora	82.38
Austin	48.09
Birmingham	1.64
Boston	397.21
Buffalo	337.35
Buffalo	386.56
Chicago	364.53
Chicago	431.73
Cincinnati	129.6
Cincinnati	1.41
Cincinnati	288.32
Cincinnati	443.78
Corpus Christi	157.91
Dallas	145.63
Fremont	404.17
Gilbert	11.31
Glendale	14.09
Indianapolis	152.77
Indianapolis	464.36
Irvine	15.19
Jersey City	369.07
Las Vegas	208.97
Los	164.5
Louisville	213.64
Lubbock	27.68
```

```
Memphis	354.44
Mesa	13.79
Miami	154.64
Miami	84.11
Newark	410.37
New York	221.35
Pittsburgh	498.29
Plano	4.65
Raleigh	61.22
Riverside	349.41
Rochester	342.62
Rochester	460.39
Rochester	485.71
San Bernardino	332.43
San Francisco	388.3
San Jose	492.8
Santa Ana	2.75
Santa Ana	282.13
Scottsdale	214.32
Stockton	180.61
Tampa	353.23
Tucson	489.93
Washington	481.31
Wichita	158.25
student@dslab:~/vb/lab5$
```

COMMAND-3

```
student@dslab:~/vb/lab5$ cat example.txt | python3 itemmap_expensive.py |
sort | python3 itemred_expensive.py
```

OUTPUT-3 (Select the maximum values at each location and display)

```
Atlanta         189.22
Aurora          82.38
Austin          48.09
Birmingham              1.64
Boston          397.21
Buffalo         386.56
Chicago         431.73
Cincinnati      443.78
Corpus Christi          157.91
Dallas          145.63
Fremont         404.17
Gilbert         11.31
Glendale        14.09
Indianapolis            464.36
Irvine  15.19
Jersey City     369.07
Las Vegas       208.97
Los     164.5
Louisville      213.64
Lubbock         27.68
Memphis         354.44
Mesa    13.79
Miami           154.64
Newark          410.37
New York        221.35
```

```
Pittsburgh	498.29
Plano	4.65
Raleigh	61.22
Riverside	349.41
Rochester	485.71
San Bernardino	332.43
San Francisco	388.3
San Jose	492.8
Santa Ana	282.13
Scottsdale	214.32
Stockton	180.61
Tampa	353.23
Tucson	489.93
Washington	481.31
Wichita	158.25
student@dslab:~/vb/lab5$
```

Reference:

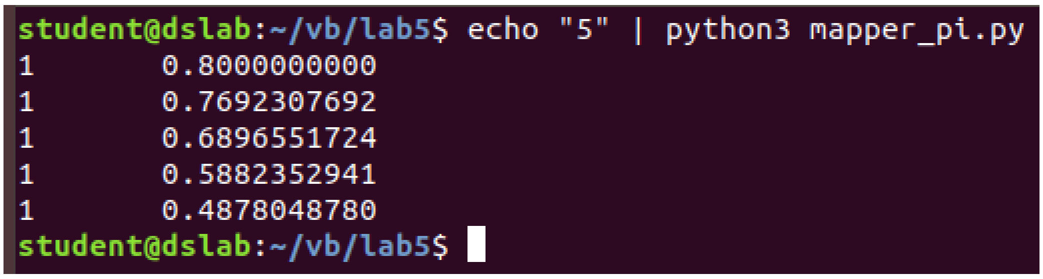
https://www-xknote-com.translate.goog/ask/a0_OPQ0XNT.html?_x_tr_sl=zh-CN&_x_tr_tl=en&_x_tr_hl=en&_x_tr_pto=sc

6. Write a mapreduce program to evaluate the PI.

mapper_pi.py

```
import sys

def f( x ):
    return 4.0 / ( 1.0 + x*x )

# input comes from STDIN (standard input)
for line in sys.stdin:

    # remove leading and trailing whitespace
    line = line.strip()

    # split the line into words
    words = line.split()
    N = int( words[0])
    deltaX = 1.0 / N

    for i in range( 0, N ):
        print("1\t%1.10f" % ( f( i * deltaX )*deltaX ) )
```

COMMAND and OUTPUT

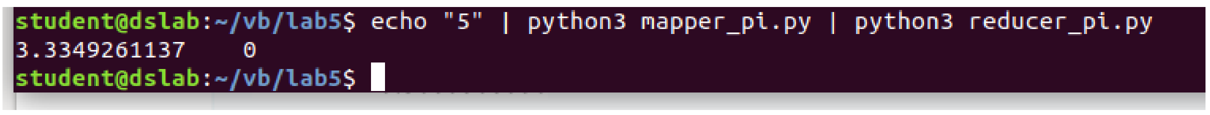


```
student@dslab:~/vb/lab5$ echo "5" | python3 mapper_pi.py
1       0.8000000000
1       0.7692307692
1       0.6896551724
1       0.5882352941
1       0.4878048780
student@dslab:~/vb/lab5$
```

reducer_pi.py

```
from __future__ import print_function
from operator import itemgetter
import sys
sum = 0
# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = float(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        #print( "--skipping (%s, %s)" % ( str(word), str(count) ) )
        continue

    sum += count

# do not forget to output the last word if needed!
print( '%1.10f\t0' % sum )
```

COMMAND and OUTPUT



```
student@dslab:~/vb/lab5$ echo "5" | python3 mapper_pi.py | python3 reducer_pi.py
3.3349261137    0
student@dslab:~/vb/lab5$ 
```