

Python MapReduce code

- We use **HadoopStreaming**
- It helps us by passing data between our Map and Reduce code via STDIN (standard input) and STDOUT (standard output)
- We use python's `sys.stdin` to read input data and print our own output to `sys.stdout`
- That's all we need to do because HadoopStreaming will take care of everything else!

Hadoop streaming

- Basically, write some code that runs like :

```
$ cat data | mapper.py | sort | reducer.py
```

Ex. 1: wordcount

- ⦿ We write a simple **MapReduce** program for Hadoop in **Python**
- ⦿ Our program reads text files and counts how often words occur
- ⦿ The input is text files and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab

Mapper

- ⦿ It will read data from STDIN, split it into words and output a list of lines mapping words to their (intermediate) counts to STDOUT
- ⦿ The Map script will **not** compute an (intermediate) sum of a word's occurrences
- ⦿ Instead, it will output “<word> 1” immediately – even though the <word> might occur multiple times in the input – and just let the subsequent Reduce step do the final sum count

Reducer

- It will read the results of the Mapper from STDIN, and sum the occurrences of each word to a final count, and output its results to STDOUT

pseudo-code

```
1: class MAPPER
2:   method MAP(docid  $a$ , doc  $d$ )
3:     for all term  $t \in \text{doc } d$  do
4:       EMIT(term  $t$ , count 1)

1: class REDUCER
2:   method REDUCE(term  $t$ , counts  $[c_1, c_2, \dots]$ )
3:      $sum \leftarrow 0$ 
4:     for all count  $c \in \text{counts } [c_1, c_2, \dots]$  do
5:        $sum \leftarrow sum + c$ 
6:     EMIT(term  $t$ , count  $sum$ )
```

Ex. 2: co-occurrences

- ◉ We want to build a word co-occurrence matrix from a corpus where
 - The co-occurrence matrix is a square $n \times n$ matrix where n is the number of unique words in the corpus (i.e, the vocabulary size)
 - A cell $m_{i,j}$ contains the number of times word w_i co-occurs with w_j within a specific context – a certain window of m words in our case

Pseudo-code

```
1: class MAPPER
2:   method MAP(docid  $a$ , doc  $d$ )
3:     for all term  $w \in \text{doc } d$  do
4:       for all term  $u \in \text{NEIGHBORS}(w)$  do
5:         EMIT(pair ( $w, u$ ), count 1)      ▷ Emit count for each co-occurrence

1: class REDUCER
2:   method REDUCE(pair  $p$ , counts [ $c_1, c_2, \dots$ ])
3:      $s \leftarrow 0$ 
4:     for all count  $c \in \text{counts } [c_1, c_2, \dots]$  do
5:        $s \leftarrow s + c$                   ▷ Sum co-occurrence counts
6:     EMIT(pair  $p$ , count  $s$ )
```


Mapper

- ⦿ It will read data from STDIN, split it into words and output a list of lines mapping pair of words to their (intermediate) counts to STDOUT
- ⦿ The Map script will **not** compute an (intermediate) sum of a word's co-occurrences
- ⦿ Instead, it will output “<pair> 1” immediately – even though the <pair> might occur multiple times in the input – and just let the subsequent Reduce step do the final sum count

Reducer

- It will read the results of the Mapper from STDIN, and sum the co-occurrences of each pair to a final count, and output its results to STDOUT

Run

- ⦿ `$ cat data | mapper.py | sort | reducer.py`
- ⦿ If that works, run it live
 - Put the data into hadoops Distributed File System (DFS)
 - Run hadoop
 - Read the output data in the DFS

Run Hadoop

- Stream data through these two files, saving the output back to HDFS:

```
$HADOOP_HOME/bin/hadoop jar \  
$HADOOP_HOME/hadoop-streaming.jar \  
-input input_dir \  
-output output_dir \  
-mapper mapper.py \  
-reducer reducer.py \  
-file mapper.py -file reducer.py
```

View output

- ⦿ View output files:
 - `$ hadoop dfs -ls output_dir`
- ⦿ Note multiple output files ("part-00000", "part-00001", etc)
- ⦿ View output file contents:
 - `$ hadoop dfs -cat output_dir/part-00000`