

Word Count exercises

1. Simple word count and use of MR counters
2. Word count and word length output
3. Filter words of size ≥ 4
4. Find word length statistics (chaining of multiple MRs)
5. Finding correct average word length

Exercise with a new text file

this is my first line rkmveri

rkmveri this makes it

hardly any relevant here

we have it here rkmveri again rkmveri very relevant

this line is full of aeae

this line has another aeae

Using these text lines in a file to do the following:

1. Find the list of line having the relevant word and count the number of relevant lines (show both the count of relevant and non-relevant lines)
2. Count the number of 'a' and 'e' in each word and list the set of unique words and their length, display sorted in ascending order of number of relevant characters in the word

The output could look as:

```
null    [[0, ["first", 1]], [0, ["full", 1]], [0, ["is", 3]], [0, ["it", 2]],
          [0, ["my", 1]], [0, ["of", 1]], [0, ["this", 4]], [1, ["any", 1]], [1, ["hardly", 1]],
          [1, ["has", 1]], [1, ["line", 3]], [1, ["rkmveri", 4]], [1, ["very", 1]], [1, ["we", 1]],
          [2, ["again", 1]], [2, ["another", 1]], [2, ["have", 1]], [2, ["here", 2]],
          [2, ["makes", 1]], [3, ["relevant", 2]], [4, ["aeae", 3]], [4, ["everywhere", 1]]]
```

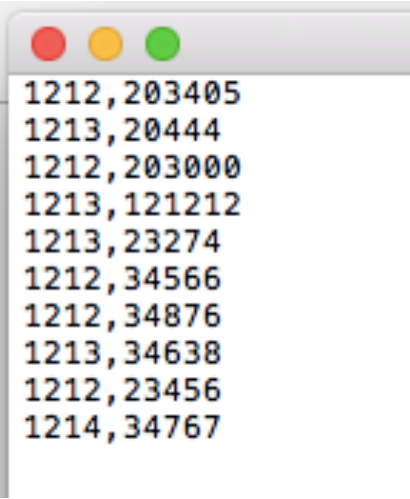
Or as:

```
2  ["have", 1], ["here", 2], ["makes", 1], ["again", 1], ["another", 1]]
3  ["relevant", 2]]
4  ["everywhere", 1], ["aeae", 3]]
1  ["very", 1], ["has", 1], ["line", 3], ["rkmveri", 4], ["any", 1], ["hardly", 1], ["we", 1]]
0  ["this", 4], ["is", 3], ["it", 2], ["my", 1], ["of", 1], ["first", 1], ["full", 1]]
```

3. Can you sort the output on each line in alphabetical order

Shopping Cart Analysis

- CustomerID, ProductID
- Exercises
 - count number of products group by custID (1)
 - create shopping basket for each custID (2)
 - create shopping basket for each custID and sort the baskets by productID (3)
 - create shopping basket for each custID and sort, invert and format the output using chain of MR and also map name to custID using helper data file, example output shown below (4)
 - find the custID who has purchased max number of products (5)
 - Does your program show more than one output if there are ties in the top rank (6)

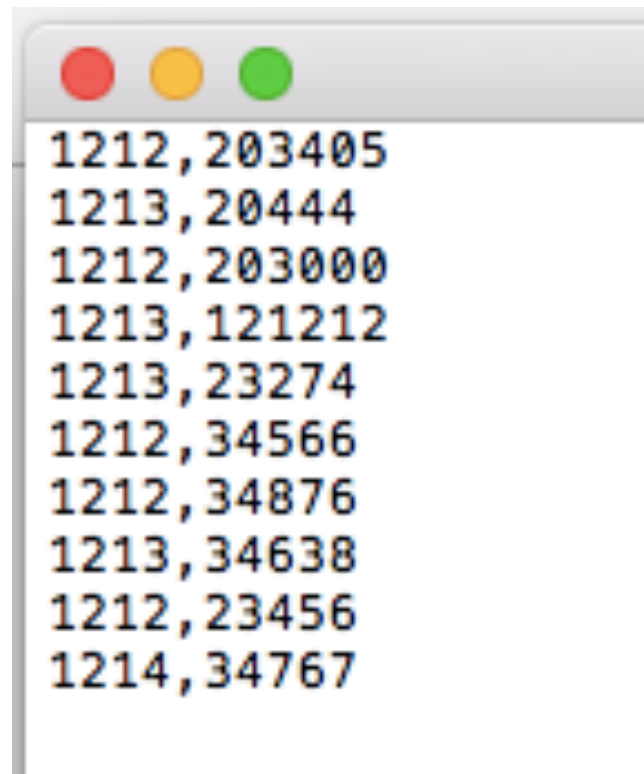


```
1212,203405
1213,20444
1212,203000
1213,121212
1213,23274
1212,34566
1212,34876
1213,34638
1212,23456
1214,34767
```

```
'Rajeev', 23456, 34566, 34876, 203000, 203405
'Pradeep', 20444, 23274, 34638, 121212
'Ravi', 34767
```

Shopping Cart Analysis

- create a text file with the following entries - sc.txt
- Create a python map reduce code file - shopping_basket1.py
- `python shopping_basket1.py mytextfile > output1`
- `cat output1`



A terminal window with a grey title bar and three colored window control buttons (red, yellow, green) on the left. The terminal displays the output of the 'cat output1' command, showing ten lines of comma-separated integer pairs. The first number in each pair represents a user ID and the second represents a product ID.

```
1212,203405
1213,20444
1212,203000
1213,121212
1213,23274
1212,34566
1212,34876
1213,34638
1212,23456
1214,34767
```

```
#count number of products group by custID
#!/usr/bin/env python

from mrjob.job import MRJob

class ShoppingBasket(MRJob):
    def mapper(self, key, line):
        (userID, productID) = line.split(',')
        yield userID, 1

    def reducer(self, userID, occurrences):
        yield int(userID), sum(occurrences)

if __name__ == '__main__':
    ShoppingBasket.run()
```

Exercise

- Try including another identical yield statement in the mapper program and explain the result
- Try removing (by commenting out) the reduce stage and see the output

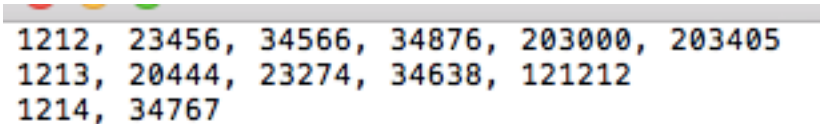
Other exercises

- shopping_basket2.py - create a shopping basket for each customer as shown below

- ```
1212 [203405, 203000, 34566, 34876, 23456]
1213 [20444, 121212, 23274, 34638]
1214 [34767]
```

# Chaining of Multiple Steps

- Create shopping basket for each custID and sort, the output result using chain of MR (shopping\_basket3.py)
- use of step function and import of MRStep module
- notice the name of the mapper and reducer functions are not standard anymore as there would be multiple such functions
- The output would look as

- 

```
1212, 23456, 34566, 34876, 203000, 203405
1213, 20444, 23274, 34638, 121212
1214, 34767
```



# Use of helper file

- create shopping basket for each custID and sort,invert and format the output result using chain of MR and also map name to custID using helper data file
- use of helper file through the use of init function reducer\_init or mapper\_init
- the init function is executed once before the first time the mapper / reducer function executes on a node
- the init function needs to be defined which reads the helper file
- configure\_options function is required to specify how the command line argument is to be provided while running the program and the help text message
- `python shopping_basket4.py --names=Cust_names.txt sc.txt > output4`

```
1212, Rajeev
1213, Pradeep
1214, Ravi
```

# Exercises

- Find the cust ID who has purchased the max number of products
- Does your program work if there are more than one customers tied for the top spot?

# Protocols in MRJob

- Default (input protocol for first step mapper is RAWValueProtocol; output from last step has JSONValueProtocol)
- RAWValueProtocol input / output format: (None, Value)
- JSONValueProtocol input / output format: (Key, Value)
- The experiments in the following slides would help you see the formats in action
- In case of default protocols no need to explicitly mention the protocols in the code otherwise need to mention the protocol name being override
- The intermediate step protocols is the internal protocol (default JSONValueProtocol) but we need not worry about it as we are never working directly with it in our code (except for the fact that we always expected (Key,Value) format as input and output in the intermediate stages (which is as it should be as it is JSONValueProtocol))

# Protocol Exercises

- Default settings (input raw, output json)
- 1212,abcd ==> "1212" "abcd"
- Now use the output file as input and use '\t' as delimiter and run again
- See the output and also look carefully at the json encoded data
- Make one of the output fields as int in the yield and look at the output now
- Again use the output file as input and use '\t' as delimiter and see output

- Now lets have custom settings (input raw, output raw) and repeat the exercises
- `from mrjob.protocol import JSONValueProtocol`
- `from mrjob.protocol import RawValueProtocol`
- `OUTPUT_PROTOCOL = RawValueProtocol`
- `yield int(userID), productID`
- `yield userID,productID`
- `yield None,userID`
- Notice lack of quotes also you cannot output list or int type (try commenting out the following lines and run)
- `#yield None,list(userID)`
- `#yield None,int(userID)`

- With default settings have the following yield statement and create an output file say out1
- `yield int(userID), productID`
- `yield userID,productID`
- Using out1 as in the input file do the following:
- Now lets have custom settings (input json, output json/raw) and repeat the exercises
- `from mrjob.protocol import JSONValueProtocol`
- `from mrjob.protocol import RawValueProtocol`
- `INPUT_PROTOCOL = JSONValueProtocol`
- (if it doesn't work use JSONProtocol and change import statement accordingly)
- Try experimenting