Hadoop Labs

Lab 1: (Checking your environment)

1. Log in to your user account on the hadoop server.
    a. Create a directory on your local C: drive. Name it Hadoop.
    b. Got to www.grut-computing.com
    c. Click on the RAS tab
    d. Open the "Support" link
    e. Under Programminbg Links, click on "Key Link"
    f. Copy the contents of this file into notepad.
        i. Save your txt file as c:\Hadoop\Hadoop.ppk
        ii. Make sure the file is NOT saved as Hadoop.ppk.txt. If it is, fix it.
    g. Open the "Hadoop Login Link". Copy the contents.
    h. Open putty.
        i. Paste the contents of the Login Link into the Host Name Link in putty.
        ii. Change "user@" to "userxx@" in the Host Name link. Put your user number instead of xx.
        iii. In the "Saved Sessions" text box, name your Session. If you can't think of a name, use "HadoopHost".
        iv. Save.
        v. Open. You should now be logged in to our online server.
        vi. By the way, your password is "hadoop666".

2. Stop here and the instructor will help you set up your putty session.

3. Look at your directory's privileges. ls -ld .

4. Look at everyone's directory privileges. ls -l /home

5. Check your alias's. alias

6. Check your .bash_profile. cat .bash_profile

7. Check your .bashrc. cat .bashrc

8. Copy the following files from /home/mark:

    a. fruit.aa, fruit.ab, fruit.ac
    b. Your command is:
        i. cp ~mark/fruit.?? . (There is a space-dot at the end of this command.)

Lab 1: (continued)

9. Type out the contents of fruit.aa — cat fruit.aa
10. Type out the contents of fruit.aa and fruit.ab. — cat fruit.aa fruit.ab
11. Type out the contents of all fruit files. — cat fruit.*
    (notice they are in order. * will match file names alphabetically.)
12. Type the contents of all fruit files with line numbers. — cat -n fruit.*
13. Now, in reverse order with line numbers. — cat -n fruit.* | tac
14. Check your home directory: — pwd
15. Check your running processes: — ps -u $LOGNAME
16. Check to ensure you can access beeline: — beeline
17. Exit beeline: — ! q
18. Check to ensure you can access pig: — pig
19. Exit pig: — quit
20. Create a directory in your home directory named scripts. mkdir scripts
21. Create a directory in your home directory named python. mkdir python
22. Look at your directory — ls -l (ell ess dash ell)
23. We'll talk about item 22.

Lab 2: (Working with HDFS)

1. Create an alias named hcat. It will be a shortcut for:
    a. hadoop fs -cat
    b. test it

2. Create the following aliases:
    a. alias hfrom='hadoop fs -copyFromLocal'
    b. alias hls='hadoop fs -ls'
    c. alias hrm='hadoop fs -rm'
    d. alias hrmdir='hadoop fs -rmdir'
    e. alias hto='hadoop fs -copyToLocal'
    f. test them all

3. When all of them are working:
    a. add the 6 aliases to your .bashrc file
    b. Type the command:
        i. . .bashrc # dot space dot bashrc
        ii. to ensure they are all syntactically correct
        iii. if not, try again

4. Bash scripting overview.
    a. Shell variables
    b. chmod

5. Write a bash script named hrmdir:
    a. It will check to see if the supplied hdfs directory exists.
    b. It will check to see if the supplied hdfs directory is owned by you.
    c. If both checks pass, the script will:
        i. delete all files in that directory
        ii. delete the directory
        iii. report back success or failure
        iv. (The instructor will help you with this one)

Lab 2: (continued)

6. piping overview

7. redirection overview

8. grep overview

9. sort overview

    a. sort -k3 -k1 # Sort ascending third field, then first field
    b. sort -r -k3 -k1 # Same as above but reverse (descending)
    c. sort -t "," -r -k3 -k1 # Same as above, comma delimited fields

10. tr overview

    a. cat file | tr "a-z" "A-Z" # convert everything to uppercase
    b. cat file | tr -d "xyz" # delete all x, y, and z
    c. cat file | tr -d -c "a-z" # delete al characters that are not a- z
    d. cat file | tr -s " " # remove duplicate spaces
    e. cat file | tr -s " " "," # remove dupe spaces and convert to comma.

11. wc overview
    a. cat file | wc -l # count lines in file
    b. cat file | wc -w # count words in file
    c. cat file | wc -c # count characters in file

12. cut overview
    a. cat file | cut -f 3,5 -d: # extract fields 3 and 5, colon delimited
    b. cat file | cut -c 1-20 # extract columns 1 thru 20

13. head and tail overview
    a. cat file | head # first 10 lines of file
    b. cat file | head -3 # first 3 lines of file
    c. cat file | tail # last 10 lines of file
    d. cat file | tail -3 # last 3 lines of file
    e. cat file | tail -n +50 # skip first 49 lines (start at line 50)

14. awk overview
    a. cat file | awk ' { print $2 ":" $3 ":" $1}' # print fields 2, 3, and 1
    b. cat file | awk -F "," '{print $2 $3 $1}' # input delimited by comma

15. sed overview
    a. cat file | sed 's/this/that/' # Change first occurrences of this to that on every line.
    b. cat file | sed 's/this/that/g' # Change all occurrences on each line
    c. cat file | sed '/this/d' # delete all lines containing regex 'this'
    d. cat file | sed -n '3,5p' # print lines 3 thru 5 only
    e. cat file | sed '1,5s/this/that/ # change this to that on lines 1 - 5 only

16. ln overview
    a. ln -s /etc/passwd my.password

17. Create a symbolic link to the instructor's file. The command is:
    a. ln -s /home/mark/data/TheConductOfLife.txt
    b. Type out the file.
    c. How many lines are in the file?
    d. Show the lines that have the word "There" on it.
    e. How many lines have the word "There" on it?
    f. How many have "There" not considering case?
    g. How many words are in the file?
    h. Copy the file into your hdfs /user directory.
    i. How many words are in the file coming from hdfs?

18. Copy the /etc/passwd file into your hdfs /user directory.
    a. List only the user names (field 1) from the passwd file in your hdfs home directory.

19. How many records are in the file /user/hadoop_data/tenyears.txt file?

20. How many records are in the file /user/hadoop_data/allyears.txt file.

21. Print the first 10 records of the /user/hadoop_data/tenyears.txt file.

22. Print the last 10 records of the /user/hadoop_data/tenyears.txt file.

23. We are going to write a script named hwc. It will mimic the behavior of the local filesystem's wc command but will work on hdfs files. The requirements are:

    a. Write output to stdout, errors and / warnings to stderr.
    b. Operate only on regular (non-directory) files.
    c. Accept zero or more command line arguments (hdfs file names or directory names)
        i. No arguments - Default to your hdfs home directory
        ii. File arguments - Default to your hdfs home directory
        iii. Directory arguments - Default to all files within that directory.
        iv. With directory arguments, do not recurse into sub-directories of that directory, i.e., don't proceed in to sub-directories of the supplied directory.
        v. Invalid arguments will not stop the program processing.
    d. Supply a maximum of 1 option (-w, -c, or -l). Default is -w.

e. Examples:
    i. hwc
        1. file1 43 words
        2. file2 1234 words
        3. file3 66 words

    ii. hwc -w
        1. ... same as prior example

    iii. hwc -l
        1. file1 6 lines
        2. file2 33 lines
        3. file3 7 lines

    iv. hwc -c /user/me
        1. file1 670 characters
        2. file2 8334 characters
        3. file3 720 characters

    v. hwc /user/me /user/you
        1. file1 43 words
        2. file2 1234 words
        3. file3 66 words
        4. yourfile1 99 words
        5. yourfile2 22 words
        6. yourfile3 799 words
        7. yourfile4 8910 words

    vi. hwc file2 file3
        1. file2 1234 words
        2. file3 66 words

    vii. hwc file3 /user/you/yourfile2 /user/you/yourfile3
        1. file3 66 words
        2. yourfile2 22 words
        3. yourfile3 799 words

    viii. hwc file2 /user/you/badfile
        1. file2 1234 words
        2. bad file: file not found

    ix. hwc file2 /user/you/badfile 2> /dev/null
        1. file2 1234 words

Lab 3: (Map Reduce)

1. We are going to build a shell script based MapReduce program that will count the unique words in a data file. The final output will be a list of all words and the number of times that word is contained in a file. We will go on the assumption that the data file may be massive and split across many servers in hdfs.

2. To start, create a shell script named **mapper.sh**. (Instructor will help you out, by the way):

    b. Our mapper.sh is going to take input from standard in, count the number of times words occur, and produce output of key-comma-value, e.g.:
        i. and,22
        ii. that,12
        iii. this,15
        iv. ...

    c. Start with:
        i. In a loop, have it read records from stdin until EOF.
        ii. Write the records to stdout unchanged.
        iii. At this point, mapper.sh will be similar to the cat command.
        iv. Test it with input coming from your hdfs' TheConductOfLife.txt file.

    d. Then, translate all spaces to newlines. Now, we have 1 word per record. (tr)

    e. Remove commas, spaces, hyphens, semi-colons,periods,and double-quotes. (tr -d)

    f. Uppercase all letters. (tr "a-z" "A-Z")

    g. Remove the blank lines (grep -v)

Lab 3: (continued)

h. Sort (sort)

i. Get a count of unique occurrences (uniq -c)

j. Remove leading spaces (sed).

k. Change the space to a tab (tr).

l. Test it with hcat /user/userxx/TheConductOfLife.txt | mapper.sh

3. Now, let's create a reducer **reducer.sh**. Reducer's job is to read the output from mapper.sh, add together all of the values with the same key, and produce our final output. Test it with:

   a. hcat /user/userxx/TheConductOfLife.txt | mapper.sh | reducer.sh

4. Recognize that hcat will reconstruct the file as a single stream of data. So, mapper.sh runs locally against the one stream of output that already has everything accumulated by key. That means that, in this simulated map-reduce environment, reducer.sh doesn't really do anything.

5. In a true map-reduce application, the mapper would be sent out to each node on the cluster, run against its slice of data, and the reducer would then have some real work to do (take all of the matching keys and reducer their accumulated values down to a single value.

6. We'll walk through and launch these two scripts as a real map-reduce application.

MapReduce with Python

1. Now let's replicate, in Python, the two shell scripts we did above.

2. Create a **mapper.py** with similar logic to mapper.sh (just use Python instead of shell). The instructor will help you.

3. Then, create a **reducer.py**. reducer.py will read from stdin, check to see if we have the same key value we had on the prior record, if so, add to an accumulator.

4. If not the same key, we will reinitialize our counter variable and continue looping.

5. Run your command sequence like this:

    a. hcat /user/userxx/TheConductOfLife.txt | mapper.py | reducer.py
    b. The instructor will then have you send your mapper.py and reducer.py as arguments to a true MapReduce application.

6. Now let's do another Python lab. In this lab we will determine the maximum temperature from a history of collected weather information. Name your mapper as **max_temperature_map.py** and your reducer as **max_temperature_reduce.py**.

7. Your mapper should produce records containing the year, a tab, and the temperature for that year.

8. Rules for the mapper's collection of data:

    a. The 4-digit year is in column position 15 thru 18.
    b. The temperature is in column position 87 thru 91. The temperature is in the form of:
        i. +0022 (positive Celsius 2.2 degrees)
        ii. -0123 (negative Celsius 12.3 degrees)
        iii. +9999 (No reading. Disregard this record)
    c. There is a quality indicator in position 92. We are only interested in quality values of 0, 1, 4, 5, or 9. Anything else would be considered an unreliable reading.

9. Test your max_temperature_map.py against file /user/hadoop_data/tenyears.txt (Years 1901 thru 1909.) When you're sure it's working, pipe its output to sort.

10. Now we will write our reducer. Reducer should select the maximum temperature for the year. Its final output will be the set of 9 key-value pairs representing the highest temperature recorded for that year.

11. The instructor will now walk you through compiling several java classes, submitting them as a MapReduce job, and viewing the output.

Lab 4: (Beeline)

1. Interactive Beeline.
    a. Start beeline.
    b. Show databases;
    c. Use pilot;
    d. show tables;
    e. How many records are in pilot_basic?
    f. How many records are in pilot_basic_orc?
    g. How many records are in pilot_basic_small?
    h. How many records are based in the state of TX?
    i. How many records total are based in either TX, OK, or NM?
    j. How many records from TX have their medical expiration in August of 2017?
    k. List the number of records by state.
    l. List the number of records by state where the pilot's medical expiration is in August of 2017.

Lab 4: (Beeline continued)

2. Command line BeeLine.

    a. Echo a query into beeline. The query returns the number of records from the pilot_basic_orc table from Texas.

    b. Build a bash script file. The script file will be named countsByState.sh. The script file will produce the count of records by State.

    c. After your countsByState.sh script is working, pipe the output into grep to extract only the records from your favorite state (probably TX)

    d. Modify your countsByState.sh script to accept an optional command line argument, a state abbreviation. If an argument is supplied select only counts from that state. Otherwise, show counts by all states.

    e. Write a query extracting from the weather_station_orc table all columns from 'TX' weather stations.

    f. Write a query extracting from the small_weather_orc table all records with good weather data.

    g. Write a query extracting from the small_weather_orc table all records from 'TX' weather stations.

    h. How many records are there from the small_weather_orc table from 'TX' recordings?

Lab 5: (Sqoop)

1. Log in to mysql:
    a. mysql -u userxx -p
    b. Enter password:

2. Look at the tables in database company.

3. Make note of the table names. Exit mysql.

4. Create a shell script named impCompany.sh.

5. impCompany.sh will import your STAFF table into your hdfs home directory. Your shell script will use a sqoop command.

6. Check your home directory to ensure you have a STAFF directory containing your table data.

7. Once your script is working, modify it. Instead of importing STAFF, have your script accept a command line argument, which will be the name of the table to import.

    a. Example: impCompany.sh STUFF

    b. If the user does not enter an argument, have the script exit with a return code of 1 and an error message of:
        i. Usage: impCompany.sh table_name [table_name ...]
    c. Otherwise, attempt the import. If the import fails, have the script produce an error message of:
        i. Invalid Table name ... STUFF does not exist
    d. If the import succeeds, have the script produce an error message of:
        i. STUFF successfully imported!

8. Once your script is working, modify it. Before importing your table, check to see if the hdfs directory of the table name exists. If so, delete the directory before attempting your import. Then, continue as before.

Lab 5: (Sqoop - continued)

9. Once your script is working, modify it. Your script will now be able to accept multiple command line arguments.

    a. Example:
        i. impCompany.sh STAFF STUFF EMPLOYEE PROJECT
    b. Example Output:
        i. STAFF successfully imported!
        ii. Invalid Table name ... STUFF does not exist
        iii. EMPLOYEE successfully imported!
        iv. PROJECT successfully imported!
    c. If all imports succeed, exit the script file with a return code of 0.
    d. Otherwise, exit with a return code of the number of imports that failed.

10. Create a new shell script named impEmpDept.sh. It will import all fields joined by the department number shared in both the Employee and Department tables. (The instructor will help on this one.)

Lab 6: (Python - Part 1)

Lab 1:

Create a python script named numtest.py. Prompt the user for the input of an integer.

Show the following:

7. The input amount squared.
8. The square root of the input amount
9. If the amount is even or odd.
10. If the amount is a small, medium, or large number (small is < 10, medium is < 50, everything else is large.

Lab 2:

Copy your numtest.py to numcompare.py. In numcompare.py prompt the user for 2 integers.

Show the following:

1. If num1 is greater than, equal to, or less than num2
2. num1 times num2
3. num1 divided by num2 yielding a truncated, integer result
4. num1 divided by num2 yielding a floating point result
5. The remainder of num1 divided by num2.

Lab 3:

Copy your numtest.py to numargs.py. Pass your value on the command line instead of prompting for the value. Test your script.

Lab 4:

Copy your numcompare.py to argscompare.py. Pass your values on the command line instead of prompting for the values. Test your script.

Lab 5:

Modify your numargs.py. Wrap your statements in a try / except block to trap non-numeric data. Terminate the script on bad input with an error message.

Lab 6:

Modify your argscompare.py. Wrap your statements in a try / except block to trap non-numeric data. Terminate the script on bad input with an error message.

Lab 7:

Create a tempconvert.py file. Your program will take two command line arguments.

The first argument will be a numeric value. The second argument will be a "C" or "F" indicating that the first argument is submitted in Fahrenheit or Celsius. If the value is submitted in Fahrenheit, convert it to Celsius and vice-versa.

Your output should look like:

> 100 degrees Celsius equals 212 Fahrenheit
>
> or
>
> 212 degrees Fahrenheit equals 100 Celsius

If the second argument is not "C" or "F", print an error and terminate the program.

If your calculation fails, jump into an except clause and terminate the program.

Example:

> Error: Second Argument must be "C" or "F"
>
> Error: Could not calculate the temp conversion of "Willie".

Lab 8:

Create a dice.py file. Every time you run the program you will be rolling two dice. The dice values will be randomly generated. The output of the program will look like this:

Die 1: 3 Die 2: 5 Dice Total: 8

Lab 9

Modify your dice.py file. Move all of your code into a function named roll_2_dice. Call your function to produce the same results.

Lab 10

Modify your dice.py file. First, modify your roll_2_dice function to accept an argument. The argument will be the number of times to roll your 2 dice.

Then, have dice.py accept an integer command line argument. The argument will be optional. If an argument is supplied, it will be a positive integer representing the number of times to roll 2 dice.

If the number is less than one, roll 2 dice 1 time. If the argument is not an integer, roll 2 dice 1 time. Otherwise, roll 2 dice the number of times requested.

Lab 11

Create a python script named upper.py. Upper.py will receive records (either by piping or file redirection) and will produce the output of the received stream of data in uppercase.

Link to the instructor's fruit file. cat the fruit file and pipe the output into upper.py. Ensure that your script is working properly.

Redirect the fruit file into your upper.py script. Ensure the script works properly.

Lab 12

Create a python script named sum.py. sum.py will one integer command line argument. If the argument does not exist or is not an integer, terminate the script with an error message.

Otherwise, add up all numbers between 1 and the number submitted and print the result. Example:

sum.py 100

5050

Lab 13

Modify your sum.py script to accommodate negative number summing. Example:

sum.py -100

-5050

Lab 14

Create add.py. add.py will add up all values supplied on the command line. Example:

add.py 10 20 30 40

Sum is 100

Lab 15

Create a python scripted named countwords.py. countwords.py will count the number of records where a word was found. The input will be a file or a stream of data. Example:

cat fruit | countwords.py ap

3 (implying that the pattern ap appeared on 3 records)

Lab 16

Create a python script named nameinfo.py. The script will take your full name as command line arguments. Example:

nameinfo.py HOMER jay SiMpSon

My name is Homer Jay Simpson.
The length of my name is 17.
My short name is H. J. Simpson.
The length of my short name is 13.
My name backwards is nospmiS .J .H.
The first space in my name is at position 5.

Lab 17:

Our data file is /home/mark/data//Baseball.csv. Look at the layout of Baseball.csv.

Create a script file named baseball.py. baseball.py should display the Date, a colon, the Model, a colon, the Orders Qty for all records. At the end a record should display the total of all Orders Qty in the form of:

Total Quantity: 1000.

Lab 18:

Modify your baseball.py script to accept a command line argument. The argument will be a 4 digit year. If the year is supplied select only those records that have are for that year. If no argument is supplied proceed as you did in the prior lab.

Lab 19:

Create a link to your baseball.py script. Name it byleather.py.

Recognize that your baseball.py script now has an additional name, byleather.py.

If the user invokes the script under the name of byleather.py, the script should require a command line argument. The argument will be the beginning of the name of the leather you want to report on.

Your output should be the Date, a colon, the Leather, and the Orders Qty for the matching leather.

You should still have your Total Quantity record displayed at the end.

Lab 20:

Our data file is /home/mark/data/fortuneCookie.txt. Look at the layout of fortuneCookie.txt.

Create a program named fortune.py. Your program should calculate the number of records in the file. Then, your program should generate a random number between 1 and the number of records in the file.

Print the record at the position returned from you random number.

Lab 21:

Notice that each record in the fortuneCookie.txt file has a character at the beginning of the record. That record indicates the fortune's category.

Have your program accept a command line argument. That argument will be the category code of the fortune records you are interested in. If the user supplies a command line category code randomly generate a fortune from only the supplied category. If the user does not supply a command line argument, proceed as you did in the prior lab.

Lab 22:

Our data file is /home/mark/data/mbox.txt. Look at the layout of mbox.txt.

Create a program named mbox.py. Read through all records. When you encounter a line that starts with "X-DSPAM-Confidence" pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence in the form of:

Average spam confidence: 0.750718518519

Lab 23:

Write a program named romeo.py. romeo.py will open the file /home/mark/data/romeo.txt and read it line by line.

For each line, split the line into a list of words using the split function.

For each word, check to see if the word is already in a list. If the word is not in the list, add it to the list.

When the program completes, sort and print the resulting words in alphabetical order as in:

['Arise', 'But', 'It', 'Juliet', 'Who', 'already', 'and', 'breaks', 'east', 'envious', 'fair', 'grief', 'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft', 'sun', 'the', 'through', 'what', 'window', 'with', 'yonder']

Lab 24:

Write a program named fromcount.py. fromcount.py will read through the mail box data and when you find a line that starts with “From”, you will split the line into words using the split function.

We are interested in who sent the message, which is the second word on the From line.

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

You will parse the From line and print out the second word for each From line, then you will also count the number of From (not From:) lines and print out a count at the end.

This is a good sample output with a few lines removed:

fromcount.py

stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu
[...some output removed...]
ray@media.berkeley.edu
cwen@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
There were 27 lines in the file with From as the first word

Lab 25:

Write a program dow.py that categorizes each mail message by which day of the week the commit was done. To do this look for lines that start with “From”, then look for the third word and keep a running count of each of the days of the week. At the end of the program print out the contents of your dictionary (order does not matter).
Sample Line:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

Sample Execution:

dow.py

{'Fri': 20, 'Thu': 6, 'Sat': 1}

Lab 26:

Write a program messagecount.py to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address, and print the dictionary.

messagecount.py

{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
'ray@media.berkeley.edu': 1}

Lab 27:

Add code to the above program to figure out who has the most messages in the file. After all the data has been read and the dictionary has been created, look through the dictionary using a maximum loop to find who has the most messages and print how many messages the person has. Print this information at the end of your dictionary dump.

zqian@umich.edu 195

Lab 28:

Create a program schoolcount.py. This program records the domain name (instead of the address) where the message was sent from instead of who the mail came from (i.e., the whole email address). At the end of the program, print out the contents of your dictionary.
schoolcount.py

{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}

Lab 29:

Modify your messagecount.py script. Revise as follows:

Read and parse the “From” lines and pull out the addresses from the line. Count the number of messages from each person using a dictionary.

After all the data has been read, print the person with the most commits by creating a list of (count, email) tuples from the dictionary. Then sort the list in reverse order and print out the person who has the most commits.

Sample Line:

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

Sample Execution:

messagecount.py

cwen@iupui.edu 5

Lab 30: This program, timeofday.py, counts the distribution of the hour of the day for each of the messages. You can pull the hour from the “From” line by finding the time string and then splitting that string into parts using the colon character.

Once you have accumulated the counts for each hour, print out the counts, one per line, sorted by hour as shown below.

Sample Execution:

timeofday.py

04 3
06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1

Lab 31:

Write a program, lettercount.py, that reads a file and prints the *letters* in decreasing order of frequency. Your program should convert all the input to lower case and only count the letters a-z.

Your program should not count spaces, digits, punctuation, or anything other than the letters a-z.

Pass your input file as an argument on the command line.

Lab 32:

Write a program, mostwords.py. This program will be invoked as follows:

```
mostwords.py  TheConductOfLife.txt   #  List of top ten most used words
mostwords.py -c 10  #  Same as prior
mostwords.py -c 50 TheConductOfLife.txt #  List of top fifty most used words
mostwords.py -a TheConductOfLife.txt  #  List of all words
```

All output will be in this form (High to low, uppercase):

```
10:A
9:THE
6:THEN
4:ARE
...
```

Lab 7: (Spark)

1. Run pyspark.

```
rdd = sc.textFile("/user/userxx/TheConductOfLife.txt")
rdd.count()
rdd.take(5)
first_five = rdd.take(5)
for record in first_five:
        print(record.upper())
quit()
```

2. Create a python spark script named readFile.py. It will create an RDD of records from TheConductOfLife.txt file in your HDFS home directory. Have it print the count of records and the first five lines. You will launch it as:
    a. spark-submit readFile.py

3. Modify your readFile.py script. Have your script default (as it is now) to TheConductOfLife.txt file. However, if you supply a command line argument, use that file instead.

    a. spark-submit readFile.py
    b. spark-submit readFile.py /user/hadoop_data/fortuneCookies.txt

4. Create a shell script named "spark.sh". It will be used to launch python-spark applications. You will launch your spark scripts like:
    a. spark.sh # Will launch the default, readFile.py
    b. spark.sh otherSparkScript.py # will launch otherSparkScript.py
    c. spark.sh otherSparkScript # will launch otherSparkScript.py
    d. Within spark.sh, reroute standard error to /dev/null

5. Create a python spark script named cookieData.py. The script will evaluate /user/hadoop_data/fortuneCookies.txt. It will produce a list of the number of fortune cookie records by category code (the category code is the first character in the file).

6. Create a python spark script named cookieData2.py. The script will produce all records that contain the word “Einstein”.

7. Modify your cookieData2.py script to produce all records that contain either “Einstein”, “Socrates”, or “Plato”. Then, total the records.

8. Modify your cookieData2.py script. After the records are printed but before the total is printed, produce the total of how many records are “Einstein” records, how many are “Socrates” records, and how many are “Plato” records.

9. Modify your cookieData2.py script. Immediately before the total is printed, print the number of records selected by category.

10. Create a python spark script named cookieData3.py. The script will produce all records that are quotes of someone (those that end with “ – somebody”). How many records are there?

11. Create a python spark script named cookieData4.py. The script will produce counts, by author, of all records that are quotes of someone. Who is quoted the most? Print only the top 5 most quoted names with their count.

12. Consider the /user/hadoop_data/HHH.100M data file:

    a. How many records are in the file?
    b. How many records have an NA age?
    c. How many records don’t have an NA age?
    d. How many riders are 40 years old?
    e. How many riders are not 40 years old?
    f. How many riders are an age of 40 thru 49?
    g. How many riders rode the 100 miles in less than 6 hours?
    h. How many riders rode the 100 miles in 6 or more hours?
    i. What was the mean time it took to complete the 100 miles?

13. Consider the following files:

    a. HHH.100M, HHH.75M, HHH.100K, HHH.50M, HHH.25M, HHH.10K
    b. How many records are in all files combined?
    c. How many records are in each file?
    d. How many males are in all files combined?
    e. how many females are in all files combined?
    f. What is the average pace for each group?
    g. What is the average page for each gender for each group?

14. Consider the following files in the /user/hadoop_data directory. The city files list the US cities with greater than 100,000 population as of 2016. The state file is a list of all states:
    a. cities2M - US cities with populations > 2 million
    b. cities1M - US cities > 1 million and <= 2 million
    c. cities500K - US cities > 500 thousand and <= 1 million
    d. citiesSmall - Rounding out the rest
    e. states - The list of 50 states
    f. stateData - The 50 states, their capital, and nickname

    Create a spark application (stateData.py) that shows:

    - The total of all cities with a population of > 100,000
    - An alphabetical list of states containing a city with > 100,000 population.
    - A list of states and how many cities they have with a population of > 100,000. Display in ascending sequence by count of cities per state.
    - A list of states that do not contain any cities with population of 100,000 or greater
    - A list of states with a population of between 250K and 750K, their capital, and their nickname. Order by population.

15. Consider the /user/hadoop_data/USPop.csv data file. Create a spark application (usPopulation.py) that answers the following questions:
    a. What is the earliest year of data?
    b. What is the latest year of data?
    c. What is the population of the earliest year of data?
    d. What is the population of the latest year of data?
    e. What is the difference between the population of the earliest year and the latest year?