Quiz 4: Hadoop for Fun and Profit (125 pts)

1. Functional Programming [25 points]

Write functional functions as described below:

1. Create a function `add` that takes an arbitrary number of arguments, and adds them all. Also create a function `sub` that subtracts all the arguments but the first from the first[1]. Also create a function `ra_sub` that performs right-associative subtraction
```
add(1, 2, 3) => 6
sub(5, 1, 2) => 2
ra_sub(5, 1, 2) => (5 - (1 - 2)) => 6
```
2. Create a function `zip` that takes an arbitrary number of sequences, and zips them, i.e. creates a list of lists, where the inner lists consist of the first elements from the given sequences, then the second elements from the given sequences, and so on.
```
zip([1, 2, 3], [4, 5, 6]) => [[1, 4], [2, 5], [3, 6]]
```

[1] Hint: The python operators are implemented in the Python operator library.

```
zip([1, 2, 3], [4, 5, 6], [7, 8, 9]) => [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

3. Create a function `zipwith` that takes a function f and an arbitrary number of sequences, and returns a list of f applied to the first elements of the given sequences, followed by f applied to the second elements of the sequences, and so on.
```
zipwith(add, [1, 2, 3], [4, 5, 6]) => [5, 7, 9]
zipwith(add, [1, 2, 3], [4, 5, 6], [1, 1, 1]) => [6, 8, 10]
```
4. Create a function `flatten` that can flatten a tree.
```
flatten([1, [2, [3, 4], [5, 6], 7], 8, [9, 10]])
    => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```
5. Create a function `group_by` that takes a function and a sequence and groups the elements of the sequence based on the result of the given function. In the example below, `len` returns the length of a sequence.
```
group_by(len, ["hi", "dog", "me", "bad", "good"])
     => {2: ["hi", "me"], 3: ["dog", "bad"], 4: ["good"]}
```

2. Confirming Hadoop Installation [15 points]

1. [3 points] Acquire the cluster.[2]
2. [3 points] Load the data into the master, move the data into HDFS.
3. [3 points] *Without writing any code of your own*, verify that you have a good installation of hadoop by running wordcount on five-books. The command is similar to
```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/singhj/five-books
/books-count
```
4. [3 points] Run wordcount using the provided **`mapper_noll.py`** and the default reducer **`aggregate`**. The command is similar to
```
mapred streaming -file ~/big-data-repo/hadoop/mapper_noll.py -mapper mapper_noll.py \
                 -input /user/singhj/five-books -reducer aggregate \
                 -output /books-stream-count
```

5. [3 points] Run wordcount using the provided **`mapper_noll.py`** and the provided reducer **`reducer_noll.py`**. The command is similar to
```
mapred streaming -files ~/big-data-repo/hadoop/mapper_noll.py ~/big-data-repo/hadoop/reducer_noll.py \
                             -mapper mapper_noll.py   \
                             -reducer reducer_noll.py \
                             -input /user/singhj/five-books \
                             -output /books-my-own-counts
```

[2] Be sure to enable the cloud-platform scope for the cluster (found under the “Manage security” menu when creating the Dataproc cluster on Compute Engine.

3. Analyzing Server Logs [55 points]

A dataset representing Apache web server logs is available as `access.log`. Each row has the following schema:

- **IP of client:** This refers to the IP address of the client that sent the request to the server.
- **Remote Log Name:** Remote name of the User performing the request. In the majority of the applications, this is confidential information and is hidden or not available.
- **User ID:** The ID of the user performing the request. In the majority of the applications, this is a piece of confidential information and is hidden or not available.
- **Date and Time:** The date and time of the request are represented in UTC format as follows: - Day/Month/Year:Hour:Minutes: Seconds +Time-Zone-Correction.
- **Request Type:** The type of request (GET, PUT, POST, etc.) that the server got. This depends on the operation that the request will do.
- **API:** The API of the website to which the request is related. Example: When a user accesses a cart on a shopping website, the API comes as /usr/cart.
- **Protocol and Version:** Protocol used for connecting with server and its version.
- **Status Code:** Status code that the server returned for the request. Eg: 404 is sent when a requested resource is not found. 200 is sent when the request was successfully served. *See the http status code registry listing for interpretations of status codes.*
- **Byte:** The amount of data in bytes that was sent back to the client.
- **Referrer:** The websites/source from where the user was directed to the current website. If none, it is represented by "-".
- **User Agent String:** The user agent string contains details of the browser and the host device (like the name, version, device type etc.).
- **Response Time:** The response time the server took to serve the request. This is the difference between the timestamps when the request was received and when the request was served.

Use Hadoop to perform analytics on the provided data[3].

1. [6+9=15 points[4]] What is the percentage of each request type (GET, PUT, POST, etc.)?
2. [6+9=15 points] What percent of the responses fall into each of the following five types?
    - Informational responses (100–199)
    - Successful responses (200–299)
    - Redirection messages (300–399)

[3] Hint: the above questions are all variants of wordcount, for example,

- #1 requires a wordcount of the various request types, divided by the total number of requests,
- #2 requires a transformation of the response code to response type before counting, and
- #3 requires the mapper to emit only those IP addresses that have a response code of 400 or greater.

[4] The construct a+b=c is taken to mean a points for the mapper, b points for the reducer and c points for the total.

- Client error responses (400–499)
- Server error responses (500–599)

3. [9+16=25 points] What 5 IP addresses generate the most client errors?

4. Presidential Speeches [15 points]

All US presidential speeches are available as a single zip in J's Github repo.

Each speech may be cleaned with this filter:

```
import requests
import re
import string
stopwords_list =
requests.get("https://gist.githubusercontent.com/rg089/35e00abf8941d72d419224cfd5b5925d
/raw/12d899b70156fd0041fa9778d657330b024b959c/stopwords.txt").content
stopwords = list(set(stopwords_list.decode().splitlines()))

def remove_stopwords(words):
    list_ = re.sub(r"[^a-zA-Z0-9]", " ", words.lower()).split()
    return [itm for itm in list_ if itm not in stopwords]

def clean_text(text):
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
    text = re.sub('[\d\n]', ' ', text)
    return ' '.join(remove_stopwords(text))
```

The goal of this analysis is to calculate the sentiment of each president's speeches. One way to compute the sentiment of a collection of words is to take the average of their valences.[5] The AFINN-165 collection contains the valences of 3,382 English words.

Write a function **`valence (text)`** such that it takes a line of any presidential speech and returns its valence after cleaning it. It should be a functional program, conforming to the pattern:

```
    def valence(text):
        return calc_valence(clean_text(text))
```

where `calc_valence(text)` is a function that you write. Be sure to test this function under any imaginable conditions, for example:

- When text is empty,
- When text is a string of non-printable characters,
- When text is a bytecode string,

[5] Source.

The function must be in a form that we can use for testing in our environment against data that you don't have access to. The presidential speeches should be considered a representative sample.

Compute the average valence of each president's speeches according to this outline:

1. [7 points] In the mapper (which is given a sequence of lines of speeches as input):
    a. Clean each line as suggested above,
    b. Calculate the valence of each word in the line,
    c. Emit a (tab-separated) key-value pair (president, word valence) for each word in the line.[6]
2. [6 points] In the reducer (which is given all (president, word valence) key-value pairs with the same key, i.e.president):
    a. Compute the average valence of all words spoken by the president,
    b. Emit a (tab-separated) key-value pair (president, sentiment of president's speeches).

[2 points] How much data, in bytes, was emitted by the mappers?

[6] The filename that contains the lines being processed by the mapper can be obtained as `os.environ['mapreduce_map_input_file']`.

5. Hadoop Errors [15 points]

When dealing with errors in Hadoop, where the execution is distributed to hundreds of workers, an error message could end up in a log file on any of those servers. This is a scavenger hunt question. We deliberately modify the code so it would occasionally fail and look for the error message so we can find them! The provided mapper for Hadoop Streaming, mapper_noll.py, with the changed lines shown in red[7].

Run Hadoop Streaming on the five books we have been using for practice, using this modified mapper. It will fail, of course!

[7 points] Where (what server & location) did the divide-by-zero error messages show up and how many did you find?

[8 points] How many such messages did you find? Is the count you found consistent with what you might expect from random.randint(0,99)?

7.:

```
#!/usr/bin/env python
import sys, re
import random
::
::
       x = 1 / random.randint(0,99)
::
::

if __name__ == "__main__":
 main(sys.argv)
```