Project Phase 1 Report:

**Guidelines on how to run the program:**

* Open the command prompt and type python3 Tokenizatio.py input\_directory\_path output\_directory\_path
* All the text files will be updated on the output directory path.

1. **Reading filenames(getfilenames):**

* Files are stored at a particular location. We take a command line input from user for input and output directory. Sys.arg helps in fetching the command-line inputs.
* This is performed by the function readFromDirectory.
* Function getfilenames reads the filenames from the directory and puts it into the list ‘Filenamelist’.

1. **Parsing the file:**

* For parsing the html files into text I have used BeautifulSoup. It removes the tags and converts the file into normal text file.

1. **Tokenize the words(tokenizewords):**

* The code iterates through the filename list thus reading file one by one.
* It parses every single file with html parser, once parsed it creates a soupfile i.e. normal text file named ‘text’.
* With text.strip() I removed any blank spaces from the file.
* With the help of nltk.RegExpTokenizer library we tokenize each word, avoiding special characters like },{,} and numbers it creates the list of words and keeps appending words into the list with the parsing of each file.
* Once done I created a separate tokenized file for every single input file to do so I used os library and os.basename function to generate the file with the same input name but the different extension i.e txt extension.

1. **Add frequency of each word :**

* Code iterates through the list of tokenizedwordlist and checks if it’s already in the dictionary named wordfrequency.
* If word is a newfound one new entry is created in the dictionary
* Else it increments the value by 1.

1. **Sorting dictionary based on words:**

* For this purpose we used dictionary and maintained word and frequency count as key value pair.
* With the sorted function I sorted the dictionary based on the alphabetical order.
* Once sorted I wrote the results in ‘Tokens.txt’ file in the given output directory.
* Later I sorted the dictionary based on frequency values of the words to do so I have used lambda function and the sorted function.
* Once done updated the results in the sortedbyfrequency file.

1. **Cleaning up the data after tokenization:**

* For this purpose, I used the RegExpTokenizer library.
* Earlier I did simple tokenization with nltk.tokenize which tokenized every possible word in the document. Since I am using nltk library I don’t need to write any regExp to segregate the significant data from insignificant one. But using the simple tokenizer resulted into following issue:
  + Tokenized numbers
  + Tokenized the punctuation marks
  + Tokenizes special characters.
* Hence, I used RegExpTokenizer in which I specified the details to avoid the special characters, numbers and punctuation marks.

1. Improvements:

* While tokenizing I got some insignificant outputs for some of the files when I checked the html file for that, I got an alert of google translate to translate the html file.
* I can improve my result set by considering these translations.
* The code currently counts the stopwords as well. I can improve that part by using stopwords provided by nltk.
* While parsing I was getting error for some files as they were cp1252 encoded for which I used errors =ignore while opening the files, which can be improvised by changing the parser type I think.

1. Mapping the estimated time.

* I am using the time library for the same. Timer starts at the beginning of every file and stops at the end of 2 for loops one for tokenizing the single file another for updating the wordfrequcency dictionary.
* I am updating time taken to parse, tokenize and enlist the words in dictionary by a single file in a list called ‘timetaken’.
* I have used matplotlib to plot the graph for input files processed in elapsed time.

1. The resultset after comparison:

* My colleague used html text parser instead of beautiful soup.
* The elapse time for the file processing was almost the same.

Output for first 50 sorted elements ( based on words):

a

aa

aaa

aaaaaamajgaaaaaacwapaaaaaaadadyaaaaaaeaaoqaae

aaaeaaiwaqaaaauaaabttvrqaaaaab

aaaeab

aaaeealktqti

aaas

aac

aachen

aacisd

aacutelt

aacutev

aads

aae

aaeaaaafaaaauku

aaeliberalis

aaemassago

aaemegvalosult

aai

aaicorp

aakb

aalen

aamc

aandalsnes

aaped

aar

aardvark

aarhus

aaron

aarp

aartselaar

aas

aasad

aascpa

aau

aauw

aazabaaaajqaaaenmt

ab

ababa

abacha

abacom

abacs

abacus

abaforum

abalczo

abalkanon

aball

abaloldalizasnak

abalon

aban

abandon

abandoned

abandoning

abandonment

abang

**Top 50 outputs of sorted by frequency:**

the : 67062

a : 57730

of : 42136

and : 34586

s : 26710

com : 24144

to : 22918

edu : 21602

in : 20708

net : 16734

az : 14168

yr : 12870

pgs : 12866

fn : 11810

is : 11578

for : 9406

on : 8864

de : 7960

es : 7506

as : 7204

that : 7116

hogy : 6784

by : 6616

an : 5958

with : 5858

or : 5666

ca : 5321

nem : 5216

be : 4622

n : 4399

k : 3972

are : 3875

us : 3744

at : 3710

it : 3652

from : 3648

this : 3616

m : 2862

magyar : 2810

not : 2551

el : 2518

t : 2498

meg : 2498

which : 2482

max : 2460

his : 2410

gov : 2396

was : 2359

ac : 2320

e : 2290

en : 2242

new : 2210

their : 2206

will : 2168

have : 2166

org : 2164

examines : 2152

analysis : 2142

egy : 2112

has : 2102

co : 2098

la : 2096

its : 2056

uk : 2038

se : 2012

no : 1990

au : 1968

ms : 1868

uu : 1860

law : 1818

all : 1810

l : 1722

state : 1720

y : 1710

ha : 1632

u : 1630

can : 1630

other : 1628

i : 1542

he : 1540

who : 1526

may : 1514

text : 1504

r : 1500

they : 1494

if : 1464

ez : 1458

most : 1438

political : 1378

more : 1362

szerint : 1350

these : 1346

how : 1340

one : 1336

c : 1330

government : 1328