Assignment 1

Title:- Write a program for Tracking Emails and Investigating Email Crimes. i.e., Write a program to analyze e–mail header

Code:-

drawbox.py:

#!/usr/bin/env python3

##############################################################################

# This file is meant to hold the functions used to print the route delays #

# in an ASCII table. #

# Copyright 2020 spcnvdr <spcnvdrr@protonmail.com> #

##############################################################################

# Print the separator between rows of data in the table

#

def print\_sep():

print("-" \* 30)

# Print the start of the table used to display hop delays

#

def print\_heading():

print("\n")

print\_sep()

print("|" + " Hop # " + "|" + " Delay (in seconds) " + "|")

print\_sep()

# Print a row in the hop delay table

# @param hop the hop number to display in the row

# @param delay the delay time to display in the row

#

def print\_row(hop, delay):

h\_format = "{0:{align}{width}}".format(hop, align="^", width=7)

d\_format = "{0:{align}{width}}".format(delay, align="^", width=20)

print("|" + h\_format + "|" + d\_format + "|")

# Print the total delay time centered in a nice format

#

def print\_total(time):

if time//60 != 0:

mins = int(time // 60)

sec = int(time % 60)

output = "Total: " + str(mins) + " min. " + str(sec) + " sec."

total = "{0:{align}{width}}".format(output, align="^", width=30)

print(total)

else:

output = "Total: " + str(time) + " sec."

print("{0:{align}{width}}".format(output, align="^", width=30))

tracemail.py:

#!/usr/bin/env python3

##############################################################################

# A simple Python 3.X.X script to analyze e-mail/s saved as plain text. #

# Copyright 2020 spcnvdr <spcnvdrr@protonmail.com> #

##############################################################################

import re

import argparse

from os import path

from dateutil import parser

from email.parser import BytesParser

from email.policy import default

from collections import deque

import drawbox

# Return the difference between 2 datetimes in seconds

# @param timea datetime object to subtract from

# @param timeb datetime object to subtract

# @returns the difference in seconds as a float

#

def time\_diff(timea, timeb):

diff = timea - timeb

return diff.total\_seconds()

# Extract the timestamp from a "Received" field

# @param A string of the received field

# @returns a datetime object of the timestamp from the Received field,

# or an error string if a format error was encountered

#

def extract\_date(recfield):

if (re.search(

r'\S{3}, {0,4} \d{1,2} \S{3} \d{1,4} \d{2}:\d{2}:\d{2} [+-]\d{4}',

recfield) is not None):

# Standard date format: Sat, 28 Dec 2019 18:11:46 -0800

reg = re.search(

r'\S{3}, {0,4} \d{1,2} \S{3} \d{1,4} \d{2}:\d{2}:\d{2} [+-]\d{4}',

recfield).group()

elif (re.search(

r'\S{3}, \d{2} \S{3} \d{1,4} \d{2}:\d{2}:\d{2}.\d{0,10} [+-]\d{4}',

recfield) is not None):

# Standard format but with fractions of a second:

# Tue, 21 Jan 2020 17:45:40.233 +0000

reg = re.search(

r'\S{3}, \d{2} \S{3} \d{1,4} \d{2}:\d{2}:\d{2}.\d{0,10} [+-]\d{4}',

recfield).group()

elif (re.search(

r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d{0,10} [+-]\d{4}',

recfield) is not None):

# Will match: '2020-01-21 17:45:40.209405196 +0000'

reg = re.search(

r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d{0,10} [+-]\d{4}',

recfield).group()

else:

print("Debug: '%s'\n" % recfield)

return "Unknown date format"

return parser.parse(reg)

# Get all the fields present in an email's headers

# @param filename the filename of an email (with header) saved as plaintext

# @returns a list of all fields found

#

def get\_fields(filename):

fields = []

# First find all the fields present in the email headers

with open(filename, "rb") as fp:

headers = BytesParser(policy=default).parse(fp)

# Add each field to a list

for j in headers:

fields.append(j + ":")

return fields

# Extract the string contained in either brackets or parentheses

# @param line The string to search

# @returns The string found, or an empty string

#

def extract\_meta(line):

ip = re.search("\[(.\*?)\]", line)

if ip:

return ip.group(0)

else:

a = re.search("\((.\*?)\)", line)

if a:

return a.group(0)

else:

return ""

# Creates a list of all the received fields present in the email's headers

# @param filename the filename of an email (with header) saved as plaintext

# @returns a list of received fields

#

def get\_received(filename):

rt = []

rec = []

tmp = ""

found = False

fields = get\_fields(filename)

# Parse the file looking for Received fields

with open(filename, "r") as fp:

for line in fp:

sep = line.split()

# Found the end of the field , add to rt list

if len(sep) != 0 and sep[0] in fields and found:

rt.append(tmp)

tmp = ""

if sep[0] != "Received:":

found = False

else:

# The next field is another Received

tmp += line

elif found:

# keep adding lines until we hit another field

tmp += line

elif "Received:" in line.split():

# Found a received field, start adding lines

tmp += line

found = True

# Format each received field into a single line and add to rec list

for j in rt:

rec.append(" ".join(j.split()))

return rec

# Extract and print the route an email took

# @param filename the filename of an email (with header) saved as plaintext

#

def print\_route(filename):

ips = deque()

names = deque()

j = 1

rec = get\_received(filename)

for k in rec:

sep = k.split()

if sep[1] == "by":

names.appendleft("Null")

ips.appendleft("None")

names.appendleft(sep[2])

ips.appendleft("")

else:

f = sep.index("from")

b = sep.index("by")

half = k.split("by")

quart = half[1].split("for")

names.appendleft(sep[f + 1])

ips.appendleft(extract\_meta(half[0]))

names.appendleft(sep[b + 1])

ips.appendleft(extract\_meta(quart[0]))

print("\n\nHop #: From --> By")

for k in range(0, len(names), 2):

print("Hop {0}: {1} {2} --> {3} {4}".format(j, names[k + 1],

ips[k + 1], names[k],

ips[k]))

j += 1

# Print time delay of each hop in the route

# @param filename the filename of an email (with header) saved as plaintext

#

def print\_delay(filename):

j = 2

total = 0.0

routes = get\_received(filename)

routes.reverse()

drawbox.print\_heading()

drawbox.print\_row("1", "\*")

drawbox.print\_sep()

for k in range(0, len(routes) - 1):

timea = extract\_date(routes[k + 1])

timeb = extract\_date(routes[k])

if type(timea) == str or type(timeb) == str:

print("{} | Invalid date".format(j))

j += 1

continue

else:

diff = time\_diff(timea, timeb)

total += diff

drawbox.print\_row(j, diff)

j += 1

drawbox.print\_sep()

drawbox.print\_total(total)

# Find a field and print it if present

# @param filename the filename of an email (with header) saved as plaintext

# @param field name of the field to print

# @param desc a description to be printed about the field

#

def print\_field(filename, field, desc):

content = "Not found"

with open(filename, "rb") as fp:

headers = BytesParser(policy=default).parse(fp)

if headers[field] is not None:

content = headers[field]

print("%s: %s" % (desc, content))

# Print basic information about an email

# @param filename the filename of an email (with header) saved as plaintext

#

def print\_basic(filename):

with open(filename, "rb") as fp:

headers = BytesParser(policy=default).parse(fp)

# Access the items from the headers dictionary:

print("To: {}".format(headers["to"]))

print("From: {}".format(headers["from"]))

print("Subject: {}".format(headers["subject"]))

print("Date: {}".format(headers["Date"]))

# Parse email and print information requested via command line arguments

# @param filename the filename of an email (with header) saved as plaintext

# @param doall if true print all the information available

# @param messageid if true print message id if present

# @param origin print originating IP address if present

# @param if true print routing information

# @param if true print user agent if present

# @param if true print delay in seconds between hops

#

def parse\_email(filename, doall, messageid, origin, route, agent, delay):

# Make sure it is a normal file and exists

if not path.exists(filename):

print("Error: The file %s was not found!" % args.FILE)

exit(1)

elif not path.isfile(filename):

print("Error: %s is not a file" % args.FILE)

exit(1)

print("\nInformation for email: %s" % filename)

print\_basic(filename)

if origin or doall:

print\_field(filename, "x-originating-ip", "Originating-IP")

if agent or doall:

print\_field(filename, "User-Agent", "Usaer Agent")

if messageid or doall:

print\_field(filename, "Message-ID", "Message-ID")

if route or doall:

print\_route(filename)

if delay or doall:

print\_delay(filename)

if \_\_name\_\_ == '\_\_main\_\_':

argp = argparse.ArgumentParser("tracemail.py", description="Analyze and "

"display "

"information "

"from e-mail "

"headers")

argp.add\_argument("-a",

"--all", help="Display everything", action="store\_true")

argp.add\_argument("-d",

"--delay", help="Print delay in seconds between hops",

action="store\_true")

argp.add\_argument("-m",

"--message\_id", help="Display message ID",

action="store\_true")

argp.add\_argument("-o",

"--origin", help="Display originating IP address",

action="store\_true")

argp.add\_argument("-r",

"--route", help="Display route information",

action="store\_true")

argp.add\_argument("-u",

"--user\_agent", help="Display user agent",

action="store\_true")

argp.add\_argument("FILE",

help="Text file/s containing an email header to analyze",

nargs="+")

args = argp.parse\_args()

for i in args.FILE:

parse\_email(i, args.all, args.message\_id, args.origin, args.route,

args.user\_agent, args.delay)

print("\_" \* 80)

Email.txt:

File with email header

Output:-

