Homework: Enron Analysis

Table of Contents

Background

For this homework assignment, you will parse through a set of 10,000 emails for the purpose of pre-analysis processing. These emails are a sample from a collection of around 500,000 emails known as the "Enron Email Dataset". This dataset contains emails sent by employees of Enron after the collapse of the company. The entire dataset can be found here: Kaggle Enron Dataset

In order to parse an email, you must first be familiar with the structure of emails. When we send emails we are not only sending heading and body content information, we are also sending metadata that describes the email structure in detail. This is known as the "header" and will contain data relating to the sender, receivers, CC information, routing information, etc. This header is formatted enough that we can use regular expressions in order to identify the information that we are interested in.

Instructions

- Your script should be named *enron.py*.
- Your script should contain two classes, Server and Email. At the end of the script should be an `if __name__ == "__main__":` statement. Specifications for each of these required program elements are given below. You may write additional classes, methods, and/or functions if you wish.
- The name of your files should consist exclusively of lower-case letters, numbers, and underscores, and the file extension .py. Your filename should not start with a number.
- Your script MUST contain docstrings.
- You should place the text file in the same directory as your python script.

Server class

Functionality

- This class stores the data for all emails found in the dataset.

Attributes

- emails: A list of email objects where each object corresponds to one email.

Methods

- __init__()
  - **Parameters**
    - Self
    - Path - The path to the file that we are going to read. In this case, the path we are passing in will be the `__emails_10k.txt__` file, however, this should not be hardcoded in.
  - **Functionality**
    - The init method should open the file specified by the path for reading and set the emails attribute to a list containing Email objects. Each email instance should correspond to each individual email found in the text file.
      - *HINT*: In order to do this the first step is to create a list where each element in the list is an entire email.
        - *Additional HINT*: Take a look at the file and how the emails are separated from each other. You do not need to write any regular expressions in order to complete this part. Instead, you can use methods that are built into the string class. Once you have that list you can then iterate through the list and write regular expressions that will identify each of the elements of interest in each email(message id, date, subject, sender, receiver, and body text). You must use regular expressions in order to do this.Keep in mind that each of these elements will be represented by strings. However, none of these elements should contain information that does not pertain to it. For example, the date attribute of an email class should not contain anything in it that is not a date.

Email class

Functionality

- This class stores the data related to individual email messages.

Attributes

- message_id: The message-id that is unique to each email message. Represented as a string.
- date: The date associated with each email message. This date does not need to be formatted in any way but it must contain only date or time information. Represented as a string.
- subject: The subject of each email message. Represented as a string.

- sender: The sender of each email message. Represented as a string.
- receiver: The receiver of each email message. Represented as a string.
- body: The body message of each email message. Represented as a string.

Methods

- __init__()
  - **Parameters**
    - self
    - message_id - A string representing the message_id
    - date - A string representing the date of the email.
    - subject - A string representing the subject of the email.
    - sender - A string representing the email address of the sender.
    - receiver - A string representing the email address of the receiver.
    - body - A string representing the body text of the email.
  - **Functionality**
    - Sets the parameters to the corresponding attributes.

`main()`

Parameters

- Path, a string that represents the path of the text file that will be parsed.

Functionality

- Create an instance of server-class using the path that was passed in and save that to a variable.

Returns

- An integer, the length of the `emails` attribute of that instance.

`parse_args()`

- Parse_args function
  - Parameters
    - args_list: a list of strings containing the command-line arguments for the program (when you call this program, you will pass sys.argv[1:] as the argument for this parameter)
  - Functionality
    - Create an instance of the ArgumentParser class from the argparse module
    - Use the add_argument() method of your ArgumentParser instance to add the following arguments:
      - Required arguments (i.e., the user can't run the program without specifying these):

        - the path to the text file (as a str)
    - *NOTE*
        - You will need to import argparse and sys, or at least parts of these modules, in order for this function to work properly. Remember, import statements belong at the top of your script, right after the script docstring.
  - Returns
    - The ArgumentParser object created.

`if __name__ == "__main__":`

- pass sys.argv[1:] to parse_args() and store the result in a variable
- call the main() function; pass in the path using the values extracted from the command line arguments by your parse_args() functionFor ease of use make sure that the text file is in the same directory as your script.

Template

```
import re

# replace this comment with your implementation of
# Email class, Server class,
# main(), argparse()

if __name__ == "__main__":
    pass
    ####You will need to write some code here
```

Running your program

Your program is designed to run from the terminal. To run it, open a terminal and ensure you are in the directory where your script is saved.

The program takes at least one command-line argument: the name of the text files (which are books) that you want to process using your script. Below are some examples of how to use the program. The examples assume you are using a Unix based OS (macOS, Linux) and your program is called `enron.py`. If you are using Windows, replace `python3` with `python`.

Basic usage

```
python3 enron.py enron_10k.txt
```