Assignment 3

due 24/3/2022

The management here at CREEPI (Cork Research Establishment for the Exploitation of Personal Information) has a client who has hired us to help investigate some dodgy corporate shenanigans.

The focus of our investigation is a trove of email from company named Enron and specifically those emails between two senior executives named Kenneth Lay and Jeffery Skilling.[1] The complete email corpus is huge: about 3GB in all with over 150 users and well over half a million individual emails. We will work with a cutdown version of a mere 89MB.

We will write a Python function that finds all of the emails sent between a group of suspects i.e. emails sent by one suspect with another suspect among the recipients. We will limit the results to those with limited circulation i.e. with 20 or fewer recipients in order to filter out blanket emails sent to all and sundry.

Write a Python function `emails_between(suspects)` that finds all emails sent between any two individuals listed in "suspects".

The parameter "suspects" is dictionary in the format shown, containing the email details of the individuals under investigation. Specifically the keys give the account/directory names and the values give a list of the email address(es) associated with that account.

```
red_flags = {
    "lay-k": ["kenneth.lay@enron.com", "klay@enron.com"],
    "skilling-j": ["skilling@enron.com", "jeff.skilling@enron.com"]
    }
```

The return type is a list of "email objects", specifically of type `email.message.Message`. The `email` class is described below.

1 Notes

1. Python has a package that facilitates the parsing of emails. To use this include the following import in your program.

   ```
   from email.parser import Parser
   ```

   The documentation for this package is available here:

   `https://docs.python.org/3/library/email.parser.html`

[1] Enron was a corporation that was a big player in the US energy sector in the 1980s and 1990s. Its collapse and bankruptcy in 2001 revealed that the company's finances had been a complete sham, deliberatly and cunningly camouflaged by the company's executives over many years. During the trial that followed, the largest fraud case in US corporate history, the email trove was introduced as evidence and so entered the public domain. Lay and Skilling were both convicted and received lengthy prison sentences and so we need not feel too squeamish about poring over their emails.

Use the "plain" `Parser` (**not** the `FeedParser`). This allows you to extract various details from an email file such as the sender, date, subject line and so on using its `parse` function.

The return of function `email.parser.parse` is an object of type `email.message.Message`.

2. A stripped down version of the Enron email corpus is available for download from the Canvas page. The top-level dircectory is named `enron_email_corpus` and contains a subdirectory named `maildir` that in turn contains the individual email directories of the various employees. In our version there are just five, including Lay (`lay-k`) and Skilling (`skilling-j`). Each employee directory in turn contains a number of subdirectories (e.g. "inbox", "sent" and so on). These subdirectories contain a collection of numbered files each of which contains the text of a single email.

3. Make sure you park your copy of this directory structure in the same directory as your Python code. **Do not hardwire absolute path names into your code**.

4. Experiment with the parsing package before tackling the whole assignment. Pluck a single email out of the collection, use the parser to read and interpret it and make sure you can extract all the relevant information (date, sender and so on).

5. For working with files and directories review the material of Lecture 7. This contains guidance on how to access a particular directory or list the contents of a directory. Again it makes sense to experiment with stepping trough the directories and individual email files before tacking the whole assignment.

6. Some of the emails may contain oddball characters that the email parser baulks at (i.e. throws an exception). Skip those emails. To prevent such emails derailing the program, use a try-except statement to wrap around the the email-parsing statement. See the slides on Exceptions in the "Bits and Pieces" section of the lectures page of the module webpage, if you need to refresh your memory of how the try statement works.

   Your submission must conform to the conditions specified below.

   (a) Code must conform tho the usual programming style strictures.

   (b) Code must strictly adhere to specified naming. The file must be named `a3.py`. The function must be named `emails_between`.

   (c) No extraneous code should be included in the file i.e. code outside the `emails_between` function apart from any necessary imports, constant definitions and helper functions. Any test code should be confined to a separate file (not to to be submitted).

   (d) Code should not include:

   - Any input statements
   - Any hard-coded file path names