Programming Exercise 15.4

Email Parser

Purpose. The purpose is for you to apply all that you have learned in your study with the book “Programming Concepts in Python.

Requirements. Write a program named **`email.py`** that opens and reads a text file and writes another text file. The purpose of the program is to extract email addresses embedded in the first (input) text file, and copy them to the second (output) text file. In computer science, this process is called "parsing". The application of the program is to make it easier to enter email addresses into an email message that is to be sent to a list of recipients, when those recipients are not already in a contacts list.

For example, college websites usually have a Faculty Directory. To send a single email message to everyone, I would have to copy/paste each email address individually from the directory into the "to", "cc", or "bcc" field of my email message. But using the new **`email.py`** program, I could save the web page containing the directory listing to a file, run the program, open its output file, and copy/paste the entire list from the file into the into the "to", "cc", or "bcc" field.

Here are the specifications for the program:

1. There are to be two console inputs to the program. For each input, there is to be a *default value*, so that the user can simply press ENTER to accept any default.

2. The two console inputs are the names of the input and output files. The default filenames are to be **`fileContainingEmails.txt`** for the input file, and **`copyPasteMyEmails.txt`** for the output file. The default location for these files is the working folder of the program. The actual names and locations of the files can be any valid filename for the operating system being used, and any existing drive and folder.

3. It is okay for the user to select input and output filenames to be the same. If the user enters another name for the input file besides the default, then the default for the *output* file should change to be the same as that of the input file. If the input and output filenames are the same, then the input file becomes replaced by the output file when the program is run.

4. The output file should be overwritten, and *not* appended to. No warning is necessary when overwriting an already-existing file.

5. Output each email to the output file, separated by a semicolon and a space. Include nothing before the first email address, and nothing after the last. Include nothing in the file besides email addresses and semicolon+space separators.

6. Do *not* allow duplicate email addresses to appear in the output file. Since you are supposed to preserve the case of email addresses, it is possible for an email address to appear more than once, cased differently -- like rburns@dvc.edu and RBurns@dvc.edu. In this case, store one of them in its originally cased form -- it does not matter which one you choose as long as it is one of them and not some other casing. (So you will have to store each email in a list as you process the input file, checking to see if it is already in the list before adding it. Then use the list to write the output file after the input file has been fully processed and closed).

7. Count the number of non-duplicate email addresses found in the input file, and list each on a separate line of console output. At the end of the list of email addresses on the console, output the total number of non-duplicate email addresses found. If the number of email addresses found is zero, do *not* write the output file for output, as that would overwrite any existing file.

8. In case an email address is split between two or more lines in the input file, ignore it. Valid email addresses must appear fully within one line of the input file. Also, each line of the input file may contain more than one email address.

9. Include friendly, helpful labels on the console output. Instead of just outputting the number of email addresses found, say something like "16 email addresses were found, and copied to the file output.txt". Or if none were found, say something like "Sorry, no email addresses were found in the file input.txt".

10. Include a message in the console output explaining to the user to open the output file and copy/paste its contents into the "to", "cc", or "bcc" field of any email message. But explain that it is best to use the "bcc" field so that everyone's email address does not appear in the message, to protect their privacy.

Hints. *Email addresses consist of the characters A-Z, a-z, 0-9, underscore, dot, hyphen, and plus.* Also, they must have exactly one `'@'` followed by, but not adjacent to, at least one `'.'`.

The procedure basically involves reading a line from a file as a text variable, and traversing the line of text to find a '@' character. If one is found, its position is saved. Then traverse backwards until an invalid email character found -- that is the position *before* the email address starts. Then traverse forwards from the '@' until an invalid email character is found -- that is the position *after* the email address ends Also count the number of '.'s found as you traverse forwards from '@' -- if any are found, then you can extract a copy of the email address as a substring. Continue from the position after the extracted email address, until no more '@'s are found.

- Read a line from the input file as `lineFromFile`. Traverse it, testing each `lineFromFile[i]` until you find a '@'. Then look *backwards* in a loop until you find an invalid email address character, or run into the start of line. Then look forwards from the same '@' in another loop until you find an invalid email address character, or run into the end of line. When looking forwards, be sure to count the number of dots ('.') found -- there needs to be at least one in a valid email address.

- Use a colllection to store found email addresses, because it's capacity does not have to be specified. Each time you find a new email address, search this list using a "Boolean Search Loop" to avoid adding duplicate addresses.

- Write a value-returning function for testing a character to see if it is a valid email address character. For example, `def isValidEmailCharacter(c):`. In it, test to see if ord(c) is >=ord("A") and <=ord("Z"), or >=ord("a") and <=ord("z"), >=ord("0") and <=ord("9"), or ==ord("."), or ==ord("-"), or ==ord("+"). If it satisfies any of these conditions, return true. Otherwise, return false.

Program I/O. Input: two filenames from the console keyboard, possibly blank to indicate the default. Output: the list of found emails, in sorted order, and an output text file.