PROGRAMMING ASSIGNMENT 4: INVOICES (50 POINTS)

Given pricing information and service usage logs, your program will generate invoices for a list of users. More specifically, this assignment uses pricing information for the use of GPT models from openai.com.

Reading: For this assignment, in addition to the material covered by the previous assignment, you need to have a good command of file operations, dictionaries and sets. Review the corresponding Handouts, textbook chapters and programming practice problems.

GETTING STARTED - DATA

Programming: Start by creating a new Python project or folder in your environment.

I will supply program data files in a **hw4_data.zip.** Unzipping this file should create two folders: **tester** and **data**. Place these folders into the folder you created for your project, where your program file will be placed. Run the starter file **hw4_checker.py** – if it produces no error message and displays a list of files, that means you project structure is ready.

Review the content of the folders with data – you will notice that each folder has:

- file named **pricelist.txt –** we will call it the *pricelist* file.
  Pricelist contains pricing records for a variety of GPT models that are priced per million of input and output tokens[1]. For example, the following line

  ```
  gpt-3.5-turbo,$1.50,$2.00
  ```

  indicates that the use of model `gpt-3.5-turbo` is charged `$1.50` for each million of input tokens, and `$2.00` for each million of output tokens.
- other files with extension **.txt**, which have prefix **log** – these are the *usage* or *log* files.
  Usage files record usage per user and model and may contain multiple records per user and per model. For example, the following line

  ```
  ylobos gpt-4-32k 12500000 5000000
  ```

  means that user `ylobos` used `gpt-4-32k` sending `12500000` input tokens, and receiving `5000000` output tokens.

Review the contents and structure these files to understand it. **tester** folder contains a smaller data set made specifically for testing, and **data** folder content includes more and larger files.

PROGRAM DESCRIPTION

Your program should perform the following functionality, illustrated by the Sample runs. In brief:

1. Ask the user for the name of the subfolder in which the data files are stored.
2. Process the pricelist file data to obtain the pricing information.
3. Process the usage files data to calculate the total quantity of input and output tokens per each user

[1] Input tokens are the pieces of text you provide to the model for it to understand what you're asking or the context of a query. Output tokens are the pieces of text the model generates as a response to your input. A token can be a word, part of a word (such as a syllable), or even punctuation.

and per each model.

4. Ask the user to enter a list of user names to generate invoices.
5. Print invoices for the users who are included in the usage files and list those users who are not, both, in alphabetical order by user name.

Let's first review a sample run, then talk more in detail about steps 1-4 and, finally, describe the functions that you will define to implement them. You must be aware of the structure and content of the data to understand how the output is generated, so make sure you have reviewed the contents and structure of **pricelist.txt** and one or two order files.

Sample run #1

```
This program will create invoices for usage of GPT models based on the pricelist and log
files of daily usage.
Please enter the name of the folder with data files or press enter for 'data': tester

Please enter the names of the user to create an invoice for separated with a
combination of commas and spaces: tpinehaven, grmmet, ylobos, lbrunter

Following users had not been using the services:
         lbrunter
         grmmet

Printing Invoices:
--------------------------------------------------------------------------------
INVOICE for user: TPINEHAVEN
For use of  gpt-4                amount due: $    75.00
For use of  gpt-4-32k            amount due: $   150.00
          ------------------------------------------------
                        Total amount due is $     225.00
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
INVOICE for user: YLOBOS
For use of  gpt-3.5              amount due: $    20.00
For use of  gpt-4-32k            amount due: $  1890.00
          ------------------------------------------------
                        Total amount due is $    1910.00
--------------------------------------------------------------------------------
```

To produce the output above, the program needs to perform the following steps:

- Read the pricelist file from the user specified data subfolder, creating a dictionary that lists pricing information for each model (functions **`readPricelist()`**,**`main()`**)
- Process usage files to create counts of input and output tokens per each user and each model stored in a dictionary (function **`aggregateLogs ()`**)
- Read user names and display those that are not included in the usage files in alphabetical order (function **`main()`**)
- Print invoices (function **`invoice()`**) for all requested users, who used the services in alphabetical order

The next section describes each function in detail. In addition, you should also define a **`tester`**() function, in which you will be placing testing code for the functions that you implement.

REQUIREMENTS

Your program should implement and test the following functions.

1. Function readPricelist(dataFolderName, fileName) (10 pts)

Define a function **`readPricelist (dataFolderName, fileName)`** to read information from the pricelist file located in the specified **`dataFolderName`** and return a dictionary storing price information as follows.

The pricelist file pr**icelist.txt** located in **tester** subfolder is included here for illustration. Each line of the pricelist file has a model name followed by the price of 1 million input and 1 million output tokens, as shown:

```
gpt-4,$30.00,$60.00
gpt-4-32k,$60.00,$120.00
gpt-3.5,$0.50,$1.50
```

The **`readPricelist`** function must be passed two parameters:

- a string naming the subfolder, where the pricelist file is stored, and
- a string naming the pricelist file.

The function must return:

- a dictionary, containing an entry for each item from the menu. The keys in the dictionary must be model names, and values must be lists consisting of two floating point numbers: price per 1M[2] input tokens and price per 1M output tokens

For example, when passed parameters `"tester", "pricelist.txt"` (file shown above), the function should return a dictionary of 3 items as follows (recall that order of keys in dictionaries does not matter):

```
{'gpt-4': [30.0, 60.0], 'gpt-4-32k': [60.0, 120.0], 'gpt-3.5': [0.5, 1.5]}
```

Hint: to compose the dictionary, go through each line of the file. Split it and add the data to the dictionary, as shown.

Test this function by calling it from function **`tester()`** with tester data, then real data.

2. Function aggregateLogs(dataFolderName, prefix = "log") (13 pts)

Define a function **`aggregateLogs(dataFolderName, prefix = "log")`** to read information from **usage files** and return a dictionary storing total token counts per user and model. The function must be passed two parameters:

- a name of a data folder, where the usage files are stored, and
- a string with the prefix, that all order file names start with, “log” is the default value.
-

The function should return a dictionary with keys corresponding to user names found in the usage files. The value for each such key is another dictionary, which has keys corresponding to the models used and values being two-element lists of input, output token totals.

[2] 1M refers to 1 million

For example, when called with parameters, **`"tester", "log",`** the function must return the following dictionary (order of keys may be different)

```
{'ylobos': {'gpt-4-32k': [15500000, 8000000],
            'gpt-3.5': [10000000, 10000000]},
'tpinehaven': {'gpt-4': [2500000, 0],
               'gpt-4-32k': [2500000, 0]},
'denglorie': {'gpt-4-32k': [125845734, 25345225],
              'gpt-4': [10000000, 5000000]}
}
```

This output reflects the content of the two usage files in the `tester` folder, content of which is shown:

File `log1563.txt`:

```
ylobos gpt-4-32k 12500000 5000000
ylobos gpt-3.5 0 5000000
tpinehaven gpt-4 2500000 0
ylobos gpt-4-32k 3000000 3000000
denglorie gpt-4-32k 125845734 25345225
```

File `log3742.txt`:

```
ylobos gpt-3.5 10000000 5000000
denglorie gpt-4 10000000 5000000
tpinehaven gpt-4-32k 2500000 0
```

To explain how the return value shown above is computed, consider the first entry in the returned dictionary, for **`ylobos`**. There are two models that **`ylobos`** has used: `gpt-4-32k` and `gpt-3.5`, which are referenced in the three lines in the first file and one in the second. Usage of `gpt-4-32k` totals to `12,500,000 + 3,000,000 = 15,500,000` input tokens, and `5,000,000 + 3,000,000 = 8,000,000` output tokens, yielding key:value pair `'gpt-4-32k': [15500000, 8000000]`. Usage of `gpt-3.5` totals to `10000000` and `10000000` input and output tokens, yielding `'gpt-3.5': [10000000, 10000000]`.

Hint: To create the dictionary, start from an empty one. For each file in the subfolder specified by dataFolderName, check if it starts with the prefix, then open the file, and read it line by line.

Each line has user name, model name, input and output token quantity.

- If the user name is not in the dictionary – you should add it, first creating a dictionary with the model name as a key and a list of input and output token quantities, respectively.
- If the user name is already in the dictionary, retrieve the dictionary associated with it. Check if the model is already in the associated dictionary, increase the associated list values by the corresponding token quantities. Otherwise, if the model is not in the dictionary, add it with the model name as a key and a list of input and output token quantities, respectively.

Test this function by calling it from function **`tester()`** with tester data , then real data.

3. Function invoice(user, pricelist, dictUsage) (10 pts)

Function **`invoice(user, pricelistDict, dictUsage)`** should print an invoice for the specified **`user`**, based on pricing information contained in **`pricelistDict`** and usage information contained in the **`dictUsage`**. These two dictionaries must be composed prior to calling this function by calling the two

functions described in points 1, 2 above.

The **`invoice`** function must print an invoice for the used models, as shown in the Sample Run. The invoice should list the models in **alphabetical** order and include total price for each mode as well as the overall total. For example, for the user ylobos described above, the invoice looks as follows (follow the shown **formatting and alignment**).

```
--------------------------------------------------------------------------------
INVOICE for user: YLOBOS
For use of  gpt-3.5              amount due: $    20.00
For use of  gpt-4-32k            amount due: $  1890.00
          ------------------------------------------------
                       Total amount due is $    1910.00
--------------------------------------------------------------------------------
```

This invoice is based on the usageDict entry for

```
'ylobos': {'gpt-4-32k': [15500000, 8000000],
           'gpt-3.5': [10000000, 10000000]}
```

and the pricing entries for the models in the **`pricelistDict`**

```
'gpt-4-32k': [60.0, 120.0]
'gpt-3.5': [0.5, 1.5]
```

For example, the amount due for `ylobos's` the use of `gpt-3.5` is computed as `10,000,000/1,000,000 * $0.5 + 10,000,000/1,000,000 * $1.5,` which yields `$20.00.`

The function should return no value. Test this function by calling it from function **`tester()`**.

4. Function main() (10 pts)

Finally, compose the main function to work as follows:

1. Ask the user to enter a folder name for the subfolder where the data is placed. If user enters nothing, use 'data' for the folder name,
2. Call the functions you already defined to create the dictionaries of prices and usage.
3. Ask the user to enter names. Compute which of these names are present among the usage dictionary keys, and print out those that are not in the usage dictionary, in **alphabetical** order.
4. For those user names found in the usage dictionary, accessing them in **alphabetical** order, call function `invoice()` to print their invoice.

Note that the most efficient way to find the difference between the keys in the usage dictionary and user-specified names, is to use python *set* operations. The use of these operations is required for full credit.

Sample run #2

This sample run uses data in the data subfolder

```
This program will create invoices for usage of GPT models based on the pricelist
and log files of daily usage.
Please enter the name of the folder with data files or press enter for 'data':
```

```
Please enter the names of the user to create   invoices for, separated with
commas and spaces: nbeavours   , aallerds

Printing Invoices:
--------------------------------------------------------------------------------
INVOICE for user: AALLERDS
For use of  gpt-3.5-turbo        amount due: $     1.06
For use of  gpt-4                amount due: $    38.95
For use of  gpt-4-32k            amount due: $    36.11
          ------------------------------------------------
                        Total amount due is $      76.12
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
INVOICE for user: NBEAVOURS
For use of  gpt-3.5              amount due: $     1.11
For use of  gpt-3.5-turbo        amount due: $     3.84
For use of  gpt-4                amount due: $    29.91
For use of  gpt-4-32k            amount due: $   122.34
          ------------------------------------------------
                        Total amount due is $     157.20
--------------------------------------------------------------------------------
```

OTHER REQUIREMENTS

1. Use style requirements from previous assignments.
2. You DO NOT need to worry about incorrect input in this assignment. Your program will be tested only with valid data.
3. Your program may be tested on other data, with varying number and names of usage files, so do not hardcode the names of usage files, their number or content. You can assume the file naming described in the Data section on page 1 of this document.

GRADING

Your program should compile without syntax errors to receive any credit. If a part of your program is working, you will receive partial credit, **but only if the program compiles without syntax errors**.

In this assignment, only valid data will be given to your program to test it.

Requirement	**Points**
Correctly implementing each function according to the specification	43
Having no code outside of function definitions except for a call to main(), and no global variables	5
Programming style: including docstring, intro comments to functions, descriptive variable names, etc.	2
Total	**50**