Scripting Project 1 – Due 2/22 11:59 pm CST

THIS IS AN EXAM. Any collaboration between students will result in academic integrity charges for all participating students. You may only use the code that I've covered in class to date and your code structure must match mine. You may not seek any outside help. The sample output video is the definitive word on what to do. Your script must run like mine and you must follow my logic. Remember that your script must run to be graded, so it's better to fully finish a function rather than halfway complete them all. You have done everything required for this project in previous tutorials and homework exercises.

Late projects will not be graded. Any script with a modification date later than the due date and time will be considered late.

There is enough flexibility in these instructions that no two scripts should be identical. Using commands not taught in this class will result in penalties. Code that is wildly different than what has been introduced in class will be assumed to be a product of AI generation and will receive no credit.

Cleanly comment your code and use white space to break up blocks of code into logical, commented sections.

Your script will be graded by running it directly in Python (not IDLE). You will use the new command os.system('cls') in your script to clear your terminal screen for usability. You will want to comment this out when testing your script in IDLE. In IDLE, you can press CTRL+h for find and replace.

Download aggregate_complaints_001.zip and aggregate_complaints_002 from the sftp server. I found it handy to make a copy of these two files in a file on my desktop. That allowed me to replace them on the desktop as needed.

CRITICAL: On Thursday, 2/23, you must download the file aggregate_complaints_003.zip from the sftp server. Your desktop should have all three complaint zip folders on the desktop along with your script clearly named "Project1.py". It doesn't matter if your Complaints folder is on the desktop or not. Failure to have this exact setup by 2/24 will result in a 10% grade penalty.

Your code layout must look like this:

Imports

Declarations

Move to the user desktop

Setup function

Unpacking function

Cleanup function

Report function

A call to the setup function

The menu inside a while True loop

Imports:

Import the libraries os, getpass, shutil, zipfile and random. You may not import any other libraries for this project.

Declarations:

Make static (capitalized) variables for each of these directories and the main file. Your code should reference these when possible rather than hard-coding entire paths:
Complaints\Zip Archives
Complaints\File Archives
Complaints\Processing
Complaints
Complaints\Complaints_Master.csv

Move to the user desktop:

This is self-explanatory

Setup Function:

1. If the Complaints folder is on the desktop remove it and everything inside it. Make the Complaints folder with the subfolders Complaints\Zip Archives and Complaints\File Archives, Complaints\Processing. Use the static variables here (no hard-coding).

2. Create the file Complaints_Master.csv and write the header. The header should include columns for Complaint ID, Date Received, Company, Product and Issue.

Unpacking Function:

1. Use listdir to create a list of all files on the desktop.
2. Iterate through that list and append any file names starting with aggregate_complaints_" into a second list. Move each zip file into the zip archives folder.
3. Show the message about processing X number of zipped files (see video)
4. Iterate through the list of active zip files that you made in step 2.
    a. This list only has the file names. Join the file name to the path for the zip archives (where you moved the zip files) to make a good zip file name.
    b. Use the zip.namelist() method to create a list of active files in the current zip file.
    c. Close your zip file
    d. Make a new list of file names from the files in the file archives folder. It will be empty when you process the first file in the first zip.
    e. Iterate through the list of active files (step 4b)
        i. If the active file (4e) isn't in the archived file list (4d) then unzip that file into the Processing folder.
            1. You should use the syntax like this:

        a. exampleZip.extract(x, path = your static variable for the processing folder). In this example, x would be the name of the iterator in step 4.3.i
    ii. Close your zip file.
5. Use listdir to make a list of file names in the processing folder.
6. Iterate through that list (step 5) to get file names
    a. Make a variable for the full path to the file name
    b. Make a variable for the full path to the file name in the File Archives folder. This is for later.
    c. Open the file to read
    d. Skip a line
    e. For line in that file:
        i. CRITICAL: If the length of that line is less than 3, break. If you don't do this the script will crash due to the last line of the JSON file being only a].
        ii. Split the line on the full colon.
        iii. Use your homework 4 skills to create clean values for the following variables:
            1. Complaint ID
            2. Date Received
            3. Company
            4. Product
            5. Issue
        iv. CRITICAL: in the middle of each record there an be an attribute for consumer comment. These can wreck our plans. So if you are trying to get an attribute that is before the comment section you can index as normal. But for an attribute that exists after (left of) the comment section you MUST use negative indexing. Ignore this at your own risk and frustration.
        v. Open the master csv file and write these values. Use the format from homework 4.
    f. Move the file to the file archives. This is where step 6b is used.
7. After iterating through all the file names print the statement about complaint file processing being done.

Cleanup Function:

1. Read all records from the csv file into a list
2. Show the message 'Removing duplicate records, please wait'
3. Show the number of current records.
    a. Right justify the test "Number of current records' within 45 spaces and concatenate the length of the list from step 1.
4. Iterate thought the list of records.
    a. See if each records is in a new list (this list is initially empty). If not, add it to the new list. This creates a list of non-duplicated records.

5. Delete the Complaints_Master.csv file
6. Make a new Complaints_Master.csv file. New header, then iterate through the list of non-duplicate records to write them to the master file.
7. Report of the number of records after removing duplicates. Justify as needed. Your output must exactly match mine.

Report Function:

1. Read all the lines from the master csv file into a new list.
2. Iterate through that list to make a new list of products.
    a. Do not include duplicates. Do this by checking each product to see if it is already in the product list. If is in that list then don't add it to the list.
3. Make a while True loop
    a. Print the available products header with the nifty line under it (actually those are dashes)
    b. Sort the product list.
    c. Iterate through the product list but use a range so we get numbers.
        i. Print the number (right justify within 2 spaces) and the correct item from the product list.
    d. Get the user choice.
    e. Error trap int(the user choice). If it fails, tell the user that it is not a valid product (see video)
    f. If the choice equals the integer zero, break.
    g. Make a list of allowable user choices by making a list from a range (that's in the slides as an example). Watch for an off-by-one error.
    h. If the user choice isn't in the allowable user choices list then tell them it's not a valid product number (see video).
    i. Continue
    j. Create a variable of the actual product choice. That would be the user choice (minus one) index of the product list.
    k. Iterate through the list of all records (step 1)
        i. Split to get the product record
        ii. If the record starts with the user choice, add that full record to a new list.
        iii. Get the issue from that record
        iv. If that issue isn't in a new record list then add it
        v. Get the company from that record
        vi. If that company isn't in a new company list then add it
    l. At this point you should have a list of issues related to the user's choice of product without duplicates, a list of companies related to the user's choice of product without duplicates and a list of full records related to the user's choice of product.
    m. Print the PRODUCT line along with the actual product name
        i. That's in the product list. You know the user's choice #. Watch for the off-by-one.

n. Print the number of companies involved. Right justify that text inside 30 spaces. The length of the company list is key here.
o. Print the number of matching records. See n.
p. Print ISSUES and the nifty underline.
q. Sort the issue list.
r. Sort the company list.
s. Iterate through the issue list to print all of the issues (see video).

A call to the setup function

1. Self explanatory

The menu inside a while True loop

2. Make a while True loop
    a. Print the menu as a single multiline string (see video)
    b. Get the user choice
    c. Each user choice should only call the appropriate function, then continue
        i. User choice 4 is just a break
        ii. User choice 5 is for everything that isn't. This handles improper choices.

What actually is this project?

> The JSON file format is very popular with data analytics and NOSQL databases. Each record includes the attribute name and the attribute value. The start of the file is a single line with just [and the end of the file is just a single line with just]. Each record is in braces {}, attribute names are in quotes (as are attribute values), the attribute name is separated from the attribute value by a colon and each attribute name / attribute value pair is separated by a comma. All of which is to say that it is a great file for parsing.
>
> Yes, there is a Python library that does this.
>
> This is real data, by the way, that I carved from a 3GB+ json file. I wrote a script for that!

Is this project error-free?

I sincerely hope so. But of course not. This is the product of 16 hours of work done this week. Corrections, changes, etc. will be posted to the Project 1 discussion board. I highly suggest that you subscribe. You are responsible for anything that I post in that discussion forum. If it requires a change in your code then yes: You have to change your code.

Good luck and have fun! It's been a fun project to create.