COMPSCI 2120/9642/DIGIHUM 2220
Assignment #4
Streaming Service Files - Using Objects

Due: Friday, December 3, 11:55pm on OWL
Weight: 20%

Learning Outcomes

By completing this assignment, you will gain/expand upon skills relating to:

- strings and csv files;
- writing and using your own classes;
- handling exceptions;
- building algorithms;
- testing your programs.

Task

In the third assignment, you were given data files, and you had to write programs to read the data from those files into a variety of data structures. The program then had to output data to new files. What if you were the one having to build and update those files in the first place, though?

It can be tedious to update data files manually, line-by-line. To improve the process, we can create an update program. This program can take a data file and an update file and produce a new data file that is an updated version of the original file.

For this assignment, you will be given files that contain information about the movies and shows that can be accessed through three different streaming services. Those files also contain subscriber information. Content and subscriber information would almost certainly not be combined in one file in the real world, and the information definitely wouldn't be stored in simple, unencrypted CSV files, but the point of the assignment is to get some experience handling updates, so we will ignore reality for now.

You will be provided with update files that identify which program and subscriber information to add, remove, or update. You will then write the updated information to a new file that contains the updated information.

Functional Requirements/Specifications

You need to create five program files:

- **`program.py`**;
- **`subscriber.py`**;
- **`streaming_service.py`**;
- **`file_processing.py`**; and
- **`main.py`**.

The `program.py` File

This file contains the constructor and methods for **`Program`** objects. These objects contain the information about a streaming service's shows and movies.

Instance Attributes

- **`title:`** the program's title, a string
- **`genre:`** the program's genre, a string
- **`creator`**: the show's creator or the movie's director, a string
- **`date`**: the release year of the movie or the show's first season, a string

Methods

- Constructor: **`__init__(self, title, genre, creator, date)`**
- Getter methods: **`get_title`**, **`get_genre`**, **`get_creator`**, **`get_release_date`**
- Setter methods: **`set_title`**, **`set_genre`**, **`set_creator`**, **`set_release_date`**
- **`__repr__(self):`** generates a string representation of a **`Program`**
  Example of output from **`__repr__:`**
  `Program("Matrix; The", "Sci-Fi", "The Wachowskis", "1999")`

A note on testing: You can test your **`Program`** objects by building **`Program`** objects at the end of the **`program.py`** file (or in a separate testing file). For example, try building the **`Program`** object that represents *The Matrix*, as in the example above.

The `subscriber.py` File

This file contains the constructor and methods for **`Subscriber`** objects. These objects contain the information about a streaming service's subscribers.

Instance Attributes

- **`name:`** the subscriber's name, a string
- **`userid:`** the subscriber's user ID, a string
- **`password`**: the subscriber's password, a string

Methods

- Constructor: **`__init__(self, name, userid, password)`**
- Getter methods: **`get_name`**, **`get_id`**, **`get_password`**
- Setter methods: **`set_name`**, **`set_id`**, **`set_password`**
- **`__repr__(self):`** generates a string representation of a **`Subscriber`**
  Example of output from **`__repr__:`**
  `Subscriber("Bill", "Ted_is_my_dude", "b3exc3llent")`

A note on testing: You can test your **`Subscriber`** objects by building **`Subscriber`** objects at the end of the **`subscriber.py`** file (or in a separate testing file). For example, try building the **`Subscriber`** object that represents Bill, as in the example above.

The `streaming_service.py` File

This file contains the constructor and methods for **`StreamingService`** objects. These objects contain the name of the streaming service, the programs it offers, and its subscriber information.

Instance Variables

- **`name:`** the streaming service name, a string
- **`programs:`** a list of **`Program`** objects belonging to the service
- **`subscribers`**: a list of **`Subscriber`** objects belonging to the service

Constructor
    `__init__(self, name, program_list, subscriber_list)`

Getter methods

Straightforward:

- **`get_name`**
- **`get_programs`**
- **`get_subscribers`**

More Complex:

- **`get_program(self, title):`** if the given program title is in the service's programs, ***returns*** the **`Program`** object with that title; if that title isn't in the service's list of programs, then the method ***returns*** **`None`**
- **`get_subscriber(self, name):`** if the given name is in the service's subscribers, ***returns*** the **`Subscriber`** object with that name; if that name isn't in the service's list of subscribers, then the method ***returns*** **`None`**

Setter methods

- **`set_name`**

Other methods

- **`add_program(self, program):`** adds the given **`Program`** object to the service's list of programs
- **`add_subscriber(self, subscriber):`** adds the given **`Subscriber`** to the service's list of subscribers
- **`remove_program(self, title):`** given a program's title, removes the **`Program`** with that title from the service's list of programs
- **`remove_subscriber(self, name):`** given a subscriber's name, removes the **`Subscriber`** with that name from the service's list of subscribers

- **`update_program(self, title, genre, creator, date):`** given a **`Program`** object's information, updates that **`Program`** object in the service's list of programs
- **`update_subscriber(self, name, userid, password):`** given a **`Subscriber`** object's information, updates that **`Subscriber`** object in the service's list of subscribers
- **`sort(self):`** sorts the service's program list by title and the service's subscriber list by name; ***returns*** a **`StreamingService`** object
  Big Hint: sorting the objects you have created isn't as simple as calling the built-in **`sorted()`** function, as that would require you to make your objects iterable. You can avoid that here by making new basic lists using program titles and subscriber names, sorting each of those lists using Python's built-in **`sort()`** function, and then instantiating a new **`StreamingService`** object. You can then use the sorted lists, along with methods from **`Program`** and **`Subscriber`**, to build the new **`StreamingService`**.

String Representation

`__repr__(self):` generates a string representation of a **`StreamingService`**

Example of output from `__repr__`:

```
StreamingService("Disney+", [Program("Guardians of the
Galaxy", "Sci-Fi", "James Gunn", "2014"),
Program("WandaVision", "Sci-Fi", "Jac Schaeffer", "2021"),
Program("Hidden Figures", "Drama", "Theodore Melfi", "2016")],
[Subscriber("Walt", "thefounder", "l33t"), Subscriber("Stan",
"thewriter", "lee1922")])
```

The `file_processing.py` File

This file contains functions that are used for processing the files associated with streaming services. It contains **three** functions. In all likelihood, you will find this to be the most difficult of the files to write. You will probably have to do a fair bit of debugging to get this working correctly.

- *Function Name:* **`build_new_service()`**

  This function reads a data file containing a streaming service's information. This function should use **`try/except`** to ensure that the file it is passed is valid.

  - *Parameter:* a data file containing streaming service data.
  - *Return Value:*
    - if the file is found, a **`StreamingService`** object built from the data file's information
    - if the file isn't found, **`None`**

  - *How it works:*

    A data file for building the **`StreamingService`** object looks like this:

```
NETFLIX
PROGRAMS
Halt and Catch Fire,Christopher Cantwell and Christopher C. Rogers,Drama,2014
Arrival,Drama,Denis Villeneuve,2016
Us,Horror,Jordan Peele,2018
Matrix; The,Sci-Fi,The Wachowskis,1999
SUBSCRIBERS
John,john123,password
Farah,f206,abcdef
```

File 1-The `netflix.csv` Data File

1. First, you will have to instantiate a **`StreamingService`** object.
2. The first line of the file is the name of the streaming service. This name will always be in uppercase letters.
3. The second line of the data file will always be PROGRAMS.
4. Everything listed after PROGRAMS and before SUBSCRIBERS will be read as a **`Program`** object. So, when you are reading this file, the information on each of these lines should be instantiated as a new **`Program`** object and stored in the streaming service's list of programs.
5. Everything after SUBSCRIBERS is a **`Subscriber`** object. The information on each of these lines will be instantiated as new **`Subscriber`** objects and stored in the streaming service's list of subscribers.

Tip: You can use **`next(reader)`** (assuming your **`csv.reader`** variable is called **`reader`**) to move to the next line (similar to using **`readline()`** with a **`.txt`** file). You may need to use this to avoid PROGRAMS and SUBSCRIBERS being read as data.

- *Function Name:* **`update_service()`**

  This function reads an update file containing changes to a streaming service's data. This function should use **`try/except`** to ensure that the file it is passed is valid and that any entries being updated already exist in the streaming service.

  - *Parameters:*
    - an update file containing streaming service data
    - a **`StreamingService`** object

  - *Return Value:*
    - if the file is found, a **`StreamingService`** object built from the data file's information
    - if the file isn't found, **`None`**
    - if an entry that requires an update isn't already present in the streaming service's lists, **`None`**

  - *How it works:*

    An update file looks like this:

```
NETFLIX
PROGRAMS
Halt and Catch Fire,Christopher Cantwell and Christopher C. Rogers,Drama,2014
Arrival,Drama,Denis Villeneuve,2016
^,Us,,,2019
-The Matrix,Science Fiction,The Wachowskis,1999
+,Squid Game,Thriller,Hwang Dong-hyuk,2021
+,Dark,Mystery,Baran bo Odar and Jantje Friese,2017
SUBSCRIBERS
-,John,john123,password
^,Farah,,abcdefg
+,Candice,candy2000,12345
```

File 2-The *`update_netflix.csv`* Data File

1. You can read this file similarly to how you read the data file, but an update file behaves differently (as you can see in Step #6).
2. The first line of the file is the name of streaming service. This name will always be in uppercase letters.
3. The second line of the update file will always be PROGRAMS.
4. Everything listed after PROGRAMS and before SUBSCRIBERS represents a **`Program`** object.
5. Everything after SUBSCRIBERS represents a **`Subscriber`** object.

6. For both the `Program` and `Subscriber` objects, you will see some lines that contain one of these symbols (`+`, `-`, `^`). What follows are instructions for dealing with lines that contain these symbols:

   `+`: this symbol means that the entry on that line is to be **added** to the appropriate streaming service list. You must instantiate a new `Program` or `Subscriber` object (depending on the section of the file that you're in), and then add it to the appropriate streaming service list.

   `-`: this symbol means that the entry on that line is to be **removed** from the appropriate streaming service list.

   `^`: this symbol means that the entry on that line requires an **update**. Notice that update lines may have entries that aren't entered (as signified by commas with nothing between/after them). These details will already be present in the object being updated, so you will only have to update the sections of information that are provided in the update file, while not overwriting the information already present in that object. For example, if the streaming service contains a `Program` with the following information:

   ```
   Program(Matrix; The,Sci-Fi,The Wachowskis,1999)
   ```

   and the update file contains the following line:

   ```
   ^,Matrix; The,Action,,
   ```

   then you should update the streaming service's program list so that it contains the following information:

   ```
   Program(Matrix; The,Action,The Wachowskis,1999)
   ```

   IMPORTANT FOR MAKING YOUR LIFE EASIER: The files that your program will be tested with, like the example files provided, will always contain the correct number of commas. Additionally, the title of a **Program** object and the name of a **Subscriber** object will never require an update.

Tip: Just as with the previous function, you can use **next(reader)** to move to the next line (similar to using **readline()** with a **.txt** file). You may need to use this to avoid PROGRAMS and SUBSCRIBERS being read as data.

Tip #2: This is one of the toughest parts of the assignment, logically speaking, so you'll want to plan this function out rather than just trying to code it immediately. You will likely need to debug it quite a bit, too.

- *Function Name:* **write_update()**

  This function writes the updated streaming service to a new file.
  - *Parameters:*
    - the name of the file to write to
    - a **StreamingService** object

  - *How it works:*

    The new file looks like this:

```
NETFLIX
PROGRAMS
Arrival,Drama,Denis Villeneuve,2016
Dark,Mystery,Baran bo Odar and Jantje Friese,2017
Halt and Catch Fire,Christopher Cantwell and Christopher C. Rogers,Drama,2014
Matrix; The,Sci-Fi,The Wachowskis,1999
Squid Game,Thriller,Hwang Dong-hyuk,2021
Us,Horror,Jordan Peele,2019
SUBSCRIBERS
Candice,candy2000,12345
Farah,f206,abcdefg
```

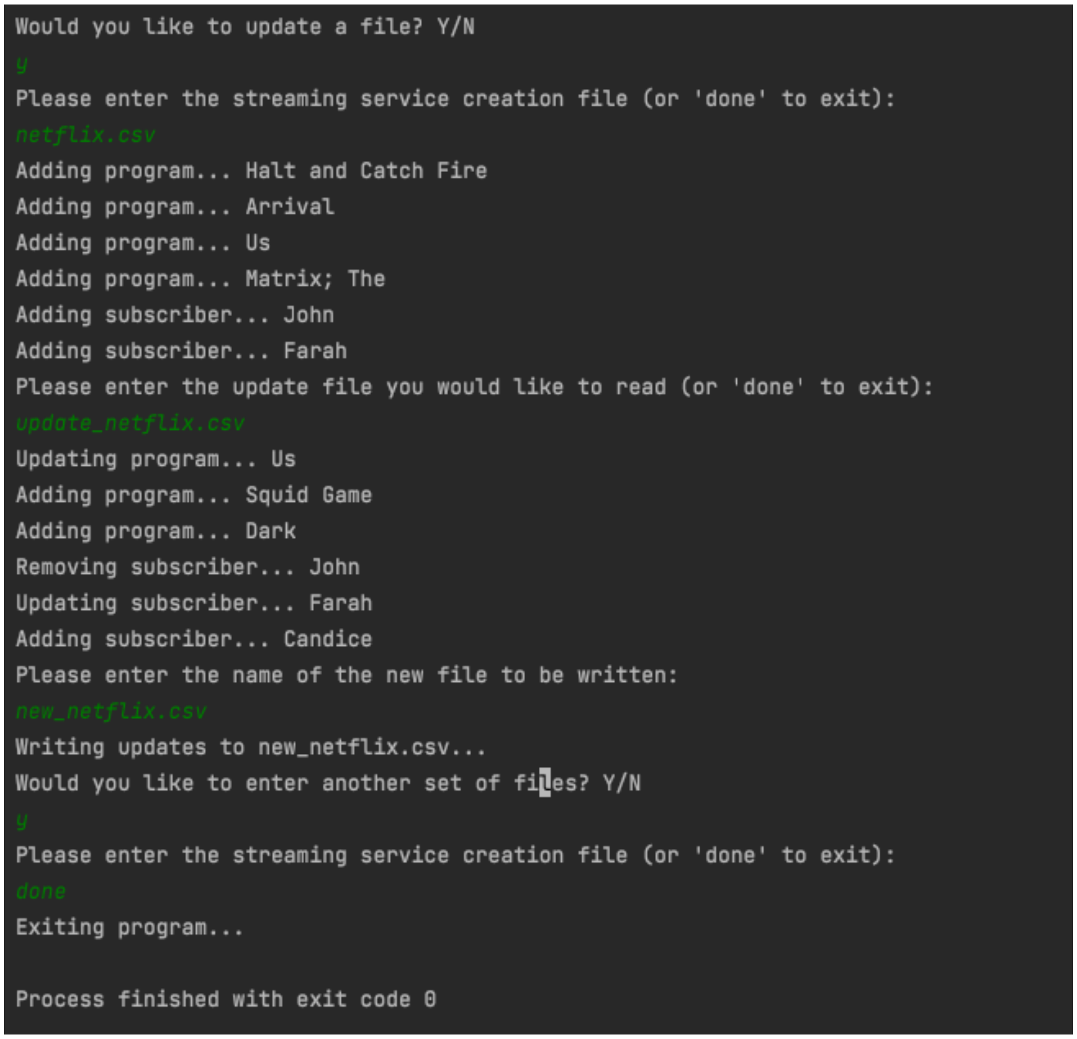
File 3-The `new_netflix.csv` Data File

1. The first line written should be the name of the service in uppercase letters.
2. The next line should be PROGRAMS.
3. The list of **Program** objects should then be written to the file. **Note:** these are not the **__repr__** versions of a **Program** object, but each of the object's elements as strings separated by commas.

4. After the programs have been written to the file, the next line should be SUBSCRIBERS.
5. The list of **`Subscriber`** objects should then be written to the file. **Note:** these are not the **`__repr__`** versions of a **`Subscriber`** object, but each of the object's elements as strings separated by commas.
   Hint for #3 and #5: If you make each row a list with the appropriate entries, the CSV writer's **`writerow()`** function should do the rest of the work.

The `main.py` File

This file contains functions the **`main()`** function that handles running the user's input. It looks like this when run:



```
Would you like to update a file? Y/N
y
Please enter the streaming service creation file (or 'done' to exit):
netflix.csv
Adding program... Halt and Catch Fire
Adding program... Arrival
Adding program... Us
Adding program... Matrix; The
Adding subscriber... John
Adding subscriber... Farah
Please enter the update file you would like to read (or 'done' to exit):
update_netflix.csv
Updating program... Us
Adding program... Squid Game
Adding program... Dark
Removing subscriber... John
Updating subscriber... Farah
Adding subscriber... Candice
Please enter the name of the new file to be written:
new_netflix.csv
Writing updates to new_netflix.csv...
Would you like to enter another set of files? Y/N
y
Please enter the streaming service creation file (or 'done' to exit):
done
Exiting program...

Process finished with exit code 0
```

`main.py`

1. The program asks the user if they want to update a file. The program accepts **`Y`** or **`y`** as 'yes' and **`N`** or **`n`** as 'no'. All other input here is considered invalid, and it should be handled using **`try/except`**. (**Hint:** raise an error if the input is invalid.) If the user chooses 'yes', then they go to Step #2. If the user chooses 'no', then the program exits.

2. The program prompts the user for a filename. The user can enter the name of a data file here, or they can type **done** to exit the program. You can have the program exit gracefully by using the function **`exit(0)`** if the user types **done**. If the user enters a filename, the program calls the **`build_new_service()`** function from **`file_processing.py`**.

3. The program prompts the user for the name of the appropriate update file. The user can enter the name of an update file here, or they can type **done** to exit the program. If the user enters an update filename, the program calls the **`update_service()`** function from **`file_processing.py`**. The result of this call should then be **sorted** so that the updated file will contain sorted lists of programs and subscribers.

4. The program prompts the user for a new filename. The user can enter the name of a file to write to, or they can type **done** to exit the program. If the user enters a new filename, the program calls the **`write_update()`** function from **`file_processing.py`**.

5. All the above steps are then repeated. The user is prompted for Y/N again, and they can build another streaming service file. At any point, they can enter **done** to exit.

6. When the user exits, any files they have created should appear in the current working directory.

Phew! You made it! It seems like a lot, and it certainly is a significant amount of programming, but most of the heavy work is in **`file_processing.py`**.

Non-Functional Requirements/Specifications

1. Include brief **comments** in your code identifying yourself (i.e., include your student number in a comment at the beginning of your program) and describing what the program does. Include **docstrings** describing what each function does. Place these comments in the appropriate places in the program.

2. Assignments are to be done individually and **must be your own work**. Software may be used to detect academic dishonesty (cheating). You may, of course, talk about your code with others and give each other tips, but you cannot share code with each other (this includes showing someone your code on a screen).

3. Use Python coding conventions and good programming techniques. Examples include:
   - meaningful variable names;
   - conventions for naming variables and classes;
   - readability; and
   - indentation.

4. Attach the **`program.py`**, **`subscriber.py`**, **`streaming_service.py`**, **`file_processing.py`**, and **`main.py`** files in OWL. **DO NOT** compress/zip them. **DO NOT** just enter code into the text submission area on OWL. You do not need to submit your output files, as the TAs will run the code and look at the output generated on their end.

5. Make sure to use **Python 3.8 or 3.9** for the interpreter. If you don't use one of these versions of Python, the tests the TAs run may fail, causing you to lose marks.

Evaluating the Assignment

The TAs will be looking at the following things when evaluating your assignment:

1. Does the program behave according to the specifications found in the assignment document?

2. Did you use objects correctly? Did you respect encapsulation, using getters and setters appropriately?

3. Does the program handle file input properly?

4. Is the file output written according to specifications? Is the new data file sorted? Is all of the data correct? Is the data written correctly, or was the **`__repr__`** method used when writing to file?

5. Does the submission meet the non-functional requirements (e.g., proper naming conventions, readability, proper file name, comments, etc.)?

The TAs will also be checking to ensure that things were not hardcoded and that your program makes use of the techniques learned in this course.

Test Cases

You have been provided with three different data files (**`netflix.csv`**, **`disney_plus.csv`**, and **`prime_video.csv`**) as well as the three update files (**`update_netflix.csv`**, **`update_disney.csv`**, and **`update_prime.csv`**) that go along with those data files.

You have also been provided with the expected output of **`main.py`** (**`new_netflix.csv`**, **`new_disney.csv`**, and **`new_prime.csv`**) for each combination of data and update files.

Finally, since you know how each file should be formatted, you should feel free to create data and update files of your own to test your program with.

Good luck!