Files and Exceptions

# Introduction

For the seventh assignment, the concept of file manipulation and exceptions was explored. This was done through a combination through learning from resources made available through the class and personal research. In addition, a new script was created to demonstrate these two concepts by modifying the task file script from previous assignments. Data was stored in **Binary files,** a type of data file that can reduce a file’s size and obscure its content albeit not encrypt it. Binary files come with many unique characteristics, most important of all is the need to pickle and unpickle them so as to preserve and interpret their data properly. If pickling is not performed, the file can not be read within a program let alone when it is opened in a text editor as its information is obscured with various meaningless symbols. Exceptions are a means of raising concern to the user that something exceptional has happened, whether that be an Python coding or input error. The user is alerted to the error through an error message that should detail what is wrong and how to fix it. Finally, I built on my experience of uploading to a Github account by creating a Github webpage that highlights the text, links and images from this word document.

# Binary Files

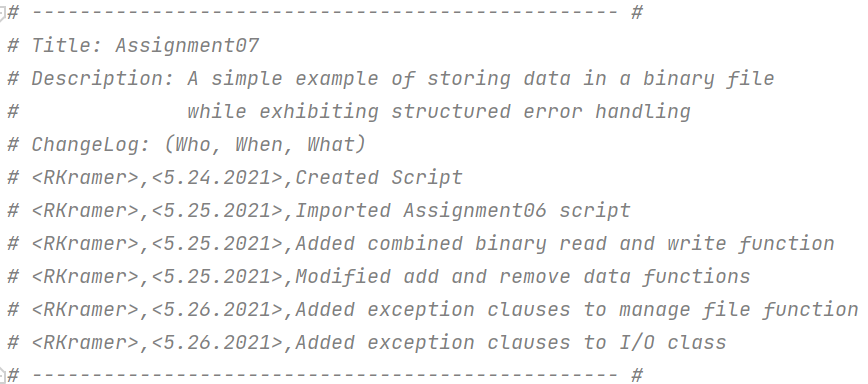
As mentioned above, **Binary Files** provide an easy way to store large data sets with less memory. They are opened to be read, written and appended much like standard text files. However once open, the data, which can range from numbers, strings, tuples, lists, dictionaries or a table combination of all sorts, is read, or rather **loaded** with the load() function, line by line through a cursor. It is best to create a loop of some sorts to ensure each line is transferred from the file to the appropriate object in the script. For uploading data to a file, the **dump()** function is employed to pickle data one by one. Another unique characteristic is that unlike text files, binary files do not convert characters to the end of the string with a ‘\n’ command. Every single character or element of binary data is represented by an 8-bit integer.

# Structured Error Handling

**Structured Handling** calls for using a **Try()** statement to section off some code that could potentially raise an exception. An **except() clause** is enacted only when any exception might be raised in the process. The **Exception class** is a built-in python class that can hold information about errors. Python automatically creates this class when an error occurs and populates it with information about the error such as its name, type and explanation. Specific exceptions can be caught when specified through naming the exception error in the except clause. Custom errors can be created with if or when statements in the try statement() that raise exceptions when a condition is met. These exceptions can be categorized under a custom exception class that is a **derived class** that “inherits” data and functions from the base class and can be modified.

# Code Outlining

My example code follows the same logic as Assignment 6’s script in that the user opens a file automatically when they run the script, albeit this time it’s a binary file. They then are prompted to either add or delete data, upload their changes to the file, reload the file or exit the program. As much of the code stayed the same, I will focus only on the areas where the use of a binary file and exceptions were found. Starting from the script header (Figure 1), one can see I imported the script from the previous assignment. I then proceeded to make an all-encompassing function for reading, writing and appending the file. Changes were made to the add and remove function functions before exception classes were added to two different function locations to make the script run smoother and with better direction for the user.



**Figure 1. Script Header**

The manage file function was the first major edit to the document as it allows the program to read, write, append and reload the binary file (Figure 2). When called upon, the mode input determines which statement to be directed to within the function. For this script, the append mode was not necessary as the user changes the list directly in the script first before appending the binary file. The overwrite mode used dump row by row to fill the file. The read function used the load function as the cursor went row by row. A for loop with a counter was used to enable all rows to be read. In addition, an exception clause was added to disregard the EOFerror that’s apparent when a for loop runs out of inputs. The reason for the error is due to the upperbound of the For loop being a thousand when there is rarely if ever a thousand inputs into the binary file. Finally, the reload mode clears the list used to store the binary file data before repeating the steps from the read mode.

@staticmethod  
**def** manage\_file(file\_name, mode, data=**None**):  
 *""" A custom wrapper function for the standard open() file function* **:param** *data: (string) with data to save* **:param** *file\_name: (string) with name of file* **:param** *mode: (string) with name of mode [Write,Overwrite,Read]* **:return***: (string) with data or write/append status  
 """* return\_data = []  
 **if** mode.lower() == **'append'**:  
 **with** open(file\_name, **"ab+"**) **as** file:  
 pickle.dump(list\_of\_rows, file)  
 return\_data = **'New data added to file!'  
 elif** mode.lower() == **'overwrite'**:  
 **with** open(file\_name, **"wb+"**) **as** file:  
 **for** row **in** list\_of\_rows:  
 pickle.dump(row, file)  
 return\_data = **'File overwritten and new data added to file!'  
 elif** mode.lower() == **'read'**:  
 **with** open(file\_name, **"rb+"**) **as** file:  
 **try**:  
 cnt = 0  
 **for** cnt **in** range(1000):  
 row = pickle.load(file)  
 list\_of\_rows.append(row) *#Load() only loads one row of data* cnt += 1  
 **except** EOFError **as** e:  
 e *#Will Always bring up error as long as table does not exceed 1000 objects* **elif** mode.lower() == **'reload'**:  
 list\_of\_rows.clear()  
 **with** open(file\_name, **"rb+"**) **as** file:  
 **try**:  
 cnt = 0  
 **for** cnt **in** range(1000):  
 row = pickle.load(file)  
 list\_of\_rows.append(row) *#Load() only loads one row of data* cnt += 1  
 **except** EOFError **as** e:  
 e *#Will Always bring up error as long as table does not exceed 1000 objects* **else**:  
 return\_data = **"No matching mode option"  
 return** return\_data

**Figure 2. Manage File function**

The adding and deleting data functions (Figure 3) required modification in that the data was stored in dictionaries within lists within a list table. Thus, the data also had to be added as a dictionary within a list to a list. For loops were required for the remove function to scan through the table for the list and then the dictionary within that list to find which dictionary and thus row needed removal based on the user’s input.

@staticmethod  
**def** add\_data\_to\_list(task, priority, list\_of\_rows):  
 *""" Adds data to a list of dictionary rows* **:param** *task: (string) with name of task:* **:param** *priority: (string) with priority level:* **:param** *list\_of\_rows: (list) you want filled with file data:* **:return***: (list) of dictionary rows  
 """* row = [{**"Task"**: str(task).strip(), **"Priority"**: str(priority).strip()}]  
 list\_of\_rows.append(row)  
 **return** list\_of\_rows, **'Success'**@staticmethod  
**def** remove\_data\_from\_list(task\_to\_remove, list\_of\_rows):  
 *""" Removes task data and associated priority from list of dictionary rows* **:param** *task\_to\_remove: (string) with name of task to be removed:* **:param** *list\_of\_rows: (list) you want filled with file data:* **:return***: (list) of dictionary rows  
 """* **for** row **in** list\_of\_rows:  
 **for** item **in** row:  
 **if** item[**"Task"**].lower() == task\_to\_remove.lower():  
 list\_of\_rows.remove(row)  
 print(**"row removed"**)  
 **return** list\_of\_rows, **'Success'**

**Figure 3. Adding and Deleting Data Functions**

A custom error was handled within the input\_new\_task\_and\_priority function (Figure 4). This was created so as to direct the user to always input one of three levels of priority. If they were to input the wrong one, there would be an error message informing them of what their input should be before asking for it again. It then would be placed into the dictionary and list with its associated task. These were the main changes implemented to improve upon this program as the main script was kept largely the same minus changing inputs for the file reading function.

@staticmethod  
**def** input\_new\_task\_and\_priority():  
 *""" Asks for new task and task's priority* **:return***: (strings) of task and priority  
 """* **try**:  
 task = str(input(**"Task:"**)).strip()  
 priority = str(input(**"Priority:"**)).strip()  
 **if** priority.lower != **"High" or "Medium" or "Low"**:  
 **raise** Exception(**'Priority must be High, Medium or Low'**)  
 **except** Exception **as** e:  
 print(**"There was an input error"**)  
 print(e)  
 print()  
 task = str(input(**"Task:"**)).strip()  
 priority = str(input(**"Priority:"**)).strip()  
 **return** task, priority

**Figure 4. Input New Task Function**

# Program Test

C:\Python\python.exe C:/\_PythonClass/Assignment07/Assignment07.py

Here is the current data from the file:

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Lawn', 'Priority': 'High'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 1

Task:Carwash

Priority:2

There was an input error

Priority must be High, Medium or Low

Task:Carwash

Priority:Low

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Lawn', 'Priority': 'High'

'Task': 'Carwash', 'Priority': 'Low'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 4

Warning: Unsaved Data Will Be Lost!

Are you sure you want to reload data from file? (y/n) - y

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Lawn', 'Priority': 'High'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 5

Goodbye!

Process finished with exit code 0

**Figure 5. Output from running Assignment07.py in pyCharm**

Microsoft Windows [Version 10.0.19041.985]

(c) Microsoft Corporation. All rights reserved.

C:\Users\robsk>CD C:\\_PythonClass\Assignment07\

C:\\_PythonClass\Assignment07>Python Assignment07.py

Here is the current data from the file:

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Lawn', 'Priority': 'High'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 1

Task:Carwash

Priority:Medium

There was an input error

Priority must be High, Medium or Low

Task:Carwash

Priority:Low

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Lawn', 'Priority': 'High'

'Task': 'Carwash', 'Priority': 'Low'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 2

Which task would you like to remove:lawn

row removed

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Carwash', 'Priority': 'Low'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Tasks To Do are: \*\*\*\*\*\*\*

'Task': 'Laundry', 'Priority': 'Low'

'Task': 'Carwash', 'Priority': 'Low'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Task

2) Remove an existing Task

3) Overwrite Data to File

4) Reload Data from File

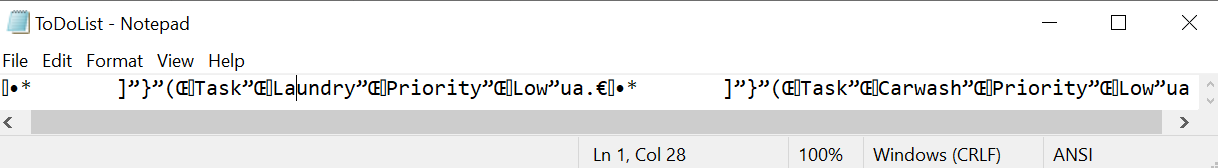
5) Exit Program

Which option would you like to perform? [1 to 5] - 5

Goodbye!

C:\\_PythonClass\Assignment07>

**Figure 6. Output from running Assignment07 in the Command Prompt**



**Figure 7. Evidence of the binary file’s save**

C:\\_PythonClass\Assignment05\ The program test was run through Pycharm (Figure 5) and through the command prompt (Figure 6). They both proved to be successful as the file was read and the program could display data, add to the list table, delete from the list table, save to the file and exit the program. A binary file was created with the appropriate data loaded (Figure 7). The data can be discerned but it is not unpickled as one would expect with it being a binary file.

# Summary

This assignment was more challenging than others in that it first required creativity to think of how to implement the error handling. It also required further research as I came across several issues with how to discern the data once it was loaded from the binary file. In particular, I had to learn how to parse through dictionaries within list tables. Finally, the github exercise was a timely reminder of proper publication of scripts and associated materials so as to be able to share that information with colleagues on the internet.