

Carolina Cerda

May 31, 2022

Foundations of Programming: Python

Assignment 07

<https://carolinacerda.github.io/IntroToProg-Python-Mod07/>

Using Error Handling, Pickling and Binary Files to Create a Script that Manages a Music Library

Introduction

In this assignment, a script that manages a mood music library playlist will be created using structured error handling, pickling and binary file to store data. This script will be able to display the user a menu of options and allow them to input an option to run one of the various functions of this program, including viewing the music library, receiving a song suggestion based on their mood, adding new moods and song suggestions, removing an existing mood and suggestion, as well as saving the data to the file and exiting the program. Additionally, this work will be posted to a new GitHub repository that includes a GitHub webpage.

Creating the Script

Structured Error Handling (Try-Except)

Errors within the code will likely appear when others use your code based on things such as if they changed the name of the data file or inputting data that doesn't match the design of the program (R. Root, "_Mod7PythonProgrammingNotes"). To provide a safety for these potential errors, it is useful to use a Try-Except block which can contain such errors within the program. A Try-Except block code can be especially useful when human interaction could be the potential cause of the issue thus within the following script for managing a mood music library this method of structured error handling will be used frequently especially within the "processing" section of the script. This concept will be expanded upon within the following sections.

Pickling

When data is saved in a binary format instead of as a plain text, this is what is known as pickling. Saving data within a binary format can be useful for reducing the file's size as it obscures the contents of the files which also makes it more difficult for humans to read. Pickling

provides the ability to serialize and deserialize objects from lists, tuples, dictionary, classes, etc to files and reconstructed back to an object (Simply Coding, [“Questions on Binary File Handling in Python”](#)). Thus, pickling, or serialization, is the process of transforming an object in memory to byte streams which can be stored as a binary file in a disk or database. Furthermore, unpickling, or deserialization, is the inverse in which a byte stream is converted back to the original object.

To store object data to a file the function `pickle.dump()` is used and has three arguments. First is the object we want stored, then the filehandle, or file object obtained by opening the target file in write-binary mode (“wb”), and lastly the protocol. To retrieve pickled data, the function `pickle.load()` is used in which a file object is created by assigning it to `pickle.load(filehandle)` in which the filehandle is the file that is opened in read-binary mode (“rb”).

To use pickle, it must be imported by using import keyword to make code in one module available in another. In the script for this assignment, this is seen at the beginning at Line 1 before anything else is done. Pickling will be revisited in the following sections located below in further detail.

```
1  import pickle
2
3  # Data ----- #
4  strFileName = 'AppData.dat'
```

PROCESSING

This section of the script is dedicated to building a class to group functions, or a group of code that performs a specific action, that will process the data within the program. “Class Processor” is a class of functions created specifically for the portion of the script that has to do with the processing code (Line 11). This is useful in that it helps to separate the processing portion of the code from the main body and overall makes the code easier to understand and delineate what is processing and what is presentation.

```
10  # Processing ----- #
11  class Processor:
12      """ Performs Processing tasks """
```

Saving Data to the File

By importing pickle, data is saved to a binary file by pickling, or serializing, the data using the `pickle.dump(object, file)` method. First, a file must be opened using the `open()` function and writing in the two parameters which are the file name, “file_name”, and the write binary mode, “wb”, that will write the data to the file (Line 23). The `pickle.dump()` method can then be written followed by closing the file using the `close()` function (Lines 24 – 25). Additionally, a try-except block is added to catch any errors that could possibly appear when running the code (Line 22, Line 26). The except block will print a message expressing to the user that there was an error and they may need to check the permissions of the file (Line 27). Finally, the “list_of_data” is returned back for calling using a return statement (Line 28).

```

14     @staticmethod
15     def save_data_to_file(file_name, list_of_data):
16         """ Saves data from a list of dictionary rows to a File
17
18         :param file_name: (string) with name of file:
19         :param list_of_data: (list) you want filled with file data:
20         :return: (list) of dictionary rows
21         """
22         try:
23             file = open(file_name, "wb")
24             pickle.dump(list_of_data, file)
25             file.close()
26         except Exception as e:
27             print("There was an error! Check file permissions.")
28         return list_of_data

```

Reading Data from the File

Since pickle was imported at the beginning of the script, as stated previously, the data can be unpickled, or deserialized, by using the `pickle.load(file)` method. First, “file” is defined by opening the file and using the mode “rb” to “read binary” (Line 38). A while loop is created which will get all of the objects in the file regardless of how many there are (Line 39). However, since an infinite loop has been created, an error might occur thus a Try-Except block is created to account for these errors (Line 40, Lines 46 – 51) (Simply Coding, [“Practice Programs on Binary File Handling in Python”](#)). Thus, the program will try to first use the `pickle.load()` method to read the binary file (Line 41). Then a for loop is created to find a line in the data and assigns the variables mood, song, and artist to a specific line in the file using a key to index (Lines 42 – 44). Then a dictionary “row” is defined using mood, song, and artist, and then “row” is appended to the `list_of_data` using the `append()` method (Lines 44 – 45).

Returning the error handling block of this code, the first except block is for an end of file error, or `EOFError`, for when all the data in the file has been read and there is no more input. To handle this error, a `break` statement is used to break the loop and continue with the rest of the program (Lines 46 – 47). For any other general error, another except block is added to capture any exception as “e” meaning that for any exception it is bound to variable “e” (Line 48). Then a message will be printed to the user expressing that there was a general error and will also print “e”, documentation of the object “e”, and class type of the argument(object) and end by breaking the loop using a `break` statement (Lines 49 – 51). Then the file is closed using `close()` method and “list_of_data” is returned using a `return` statement (Lines 52 – 53).

```

30     @staticmethod
31     def read_data_from_file(file_name, list_of_data):
32         """ Reads data from a file into a list of dictionary rows
33
34         :param file_name: (string) with name of file:
35         :param list_of_data: (list) you want filled with file data:
36         :return: (list) of dictionary rows
37         """
38         file = open(file_name, "rb")
39         while True:

```

```

40     try:
41         dict_row = pickle.load(file)
42         for line in dict_row:
43             mood, song, artist = line["Mood"], line["Song"], line["Artist"]
44             row = {"Mood": mood.strip(), "Song": song.strip(), "Artist": artist.strip()}
45             list_of_data.append(row)
46     except EOFError:
47         Break
48     except Exception as e:
49         print("There was a general error!")
50         print(e, e.__doc__, type(e), sep="\n")
51         Break
52     file.close()
53     return list_of_data

```

Recommending a Song from the Data

This function matches a mood inputted by the user to a mood in the playlist to recommend to them a corresponding song suggestion. Similar to removing data from the list, a for loop and if statement are used to find a row in the list, "list_of_data", that match each other, using the lower() method to ensure that the lower case of these values are compared, and if the value for the keyword, "mood", matches the mood that is inputted by the user then a message is returned using two print() statements where the first returns to the user the mood they inputted and the corresponding song and artist recommendation then the second wishes that the user enjoys the song then uses a break statement to break the if loop (Lines 63 – 68). An else statement is used in the if loop to ensure that the program passes through every row in the list of dictionary rows using the pass statement to avoid printing the messages multiple times (Lines 69 – 70). In the for loop, for anything else, an else statement is used to print a message expressing to the user that there was no result for their input and to try again or add more data (Lines 72-75). An extra line is added for looks then "list_of_data" is returned using the return statement (Lines 76 – 77).

```

55     @staticmethod
56     def recommend_data_from_list(mood, list_of_data):
57         """ Retrieves data from a list of dictionary rows
58
59         :param mood: (string) with name of mood:
60         :param list_of_data: (list) you want filled with file data:
61         :return: (list) of dictionary rows
62         """
63         for row in list_of_data:
64             if str(mood).lower() == str(row["Mood"]).lower():
65                 print("Since your mood today is ", mood, ", the song recommended for you is ",
66 str(row["Song"]).title(), " by ", str(row["Artist"]).title(), sep="", end=". \n")
67                 print("\nHope you like it! 🥰 😊) / ♥️")
68                 Break
69             else:
70                 Pass
71         else:
72             print("Sorry, there is no recommendation for the mood, ", mood, "! Please try again or select "

```

```

73                                     "'Option 3' to add a new "
74                                     "song recommendation for "
75                                     "this mood.", sep="")
76     print() # Add an extra line for looks
77     return list_of_data

```

Adding Data to the List

This function adds data to a list of dictionary rows. To complete this function, the dictionary, “new_row”, is created from the user input received below in the class “io” dedicated to receiving input to add new data and then it is appended to the list, “list_of_data” (Lines 89 – 90). Then the “list_of_data” is returned back to the program (Line 91).

```

79     @staticmethod
80     def add_data_to_list(mood, song, artist, list_of_data):
81         """ Adds data to a list of dictionary rows
82
83         :param mood: (string) with name of mood:
84         :param song: (string) with name of song:
85         :param artist: (string) with name of artist:
86         :param list_of_data: (list) you want filled with file data:
87         :return: (list) of dictionary rows
88         """
89         new_row = {"Mood": str(mood).strip(), "Song": str(song).strip(), "Artist": str(artist).strip()}
90         list_of_data.append(new_row)
91         return list_of_data

```

Removing Data from the List

To create a function that removes data from the list, a for loop and if statement are used to find a row in the list, “list_of_data”, that match each other, using the lower() method to ensure that the lower case of these values are compared, and if the value for the keyword, “mood”, matches the mood that is inputted by the user then the row is removed from the list using the remove() method (Lines 102 – 104). If it is a match, then a message is printed to ensure the user that the song suggestion for the mood was removed and the program breaks to end the if loop (Lines 105 – 107). An else statement is added to ensure that the input passes through every row and doesn’t print the message in the else statement for the for loop multiple times (Lines 108 – 109). To end the for loop, if no match is found, then a message will be printed to the user informing them that the mood could not be found and to try again (Lines 110 – 112). The list is then returned using the return statement that ends the functions and returns the “list_of_data” to what calls to it (Line 115). A Try-Except error handling was also added as a safety net in case the program where to run into any unforeseen errors (Line 101, Line 113 – 114).

```

93     @staticmethod
94     def remove_data_from_list(mood, list_of_data):
95         """ Removes data from a list of dictionary rows
96
97         :param mood: (string) with name of mood:

```

```

98         :param list_of_data: (list) you want filled with file data:
99         :return: (list) of dictionary rows
100         """
101     try:
102         for row in list_of_data:
103             if mood.lower() == row["Mood"].lower():
104                 list_of_data.remove(row)
105                 print(" Song suggestion for the mood, ", "", mood.title(), "", " ", was successfully removed!\n",
106                       sep="")
107                 Break
108             else:
109                 Pass
110         else:
111             print(" Sorry, the song suggestion for mood, ", "", mood.title(), "", " ", could not be found. "
112                   "Please try again.\n", sep="")
113     except:
114         print("An error has occurred!")
115     return list_of_data

```

PRESENTATION (INPUT/OUTPUT)

This section of the script is dedicated to building a class of functions that will obtain user input and present data output within the program by using a class, “IO”.

```

118     # Presentation (Input/Output) ----- #
119     class IO:
120         """ Performs Input and Output tasks """

```

Outputting Menu Tasks

A function is created that will print to the user a menu of all the options possible for this program and a brief introduction about the program. This section consists of three main print() function statements where the last adds an extra line for looks while the first two print the introduction and the menu of options (Lines 128 – 141).

```

122     @staticmethod
123     def output_menu_tasks():
124         """ Display a menu of choices to the user
125
126         :return: nothing
127         """
128         print("""
129             Welcome to the Mood Library!
130             This is a music library that recommends a song to you based on your mood.
131             Below is a menu of options from which you can choose from. Enjoy!
132             """)
133         print("
134             Menu of Options
135             1) View Current Music Library
136             2) Get a Song Suggestion Based on Your Mood
137             3) Add a Song for a Mood
138             4) Remove a Mood and its Song

```

```

139         5) Save Changes to Library
140         6) Exit Program"""
141     print() # Add an extra line for looks

```

Inputting Menu Choice

This section of code is almost identical to the starting code provided by Professor Root for Assignment 06. The only difference is expanding the number of options thus resulting in a greater range of options when asking the user for input. The change is highlighted below.

```

143     @staticmethod
144     def input_menu_choice():
145         """ Gets the menu choice from a user
146
147         :return: string
148         """
149         choice = str(input("Please choose an option [1 to 6]: ")).strip()
150         print() # Add an extra line for looks
151         return choice

```

Outputting Current Moods in List

To output the entire music library, first a try-finally block is created. A finally block always executes after a try block if the conditions are met. To ensure that if there is no data in the file a message will still print, the finally block uses an if statement to say that if the integer length of number of items in “list_of_data” is 0 (i.e., there is no data in the file) then it will print a message to the user expressing that there is no music in the mood library and to choose another option to add music (Lines 168 – 170). Before that, the program will try, if the integer length of number of items in “list_of_data” is greater than 0 (i.e., there is data in the file), then it will first print a “table” with the current moods and the suggested songs in the formatted order by using a for loop to print out each row of data in “list_of_data” (Lines 160 – 167).

```

153     @staticmethod
154     def output_current_moods_in_list(list_of_data):
155         """ Shows the current song suggestions for moods in the list of dictionaries rows
156
157         :param list_of_data: (list) of rows you want to display
158         :return: nothing
159         """
160         try:
161             if int(len(list_of_data)) > 0:
162                 print("----- YOUR MUSIC LIBRARY -----")
163                 print("MOOD | SONG SUGGESTION")
164                 for row in list_of_data:
165                     print(row["Mood"] + " | " + row["Song"] + " by " + row["Artist"])
166                 print("-----")
167                 print() # Add an extra line for looks
168             finally:
169                 if int(len(list_of_data)) == 0:
170                     print(" No current music in mood library! Please choose 'Option 3' to add songs to the library!\n")

```

Inputting a Mood to Recommend a Song

This function aims to receive user input for what mood they are feeling today and to present a recommended song back to them. Also, a list of available moods to choose from are displayed before they input their mood. If there is no music in the library then it passes over to the Processor. `recommend_data_from_list(mood, list_of_data)` function and displays the message for if there is no mood found. Using an if statement if the integer length of number items in “list_of_data” is greater than 0 then it will print a “table” of moods that are available to choose from and uses a for loop to get the row[“Mood”] value from every row in “list_of_data” (Lines 179 – 184). Then the input for the user’s mood is obtained through the `input()` function and is converted to a string ensure that the input is titled and also stripped of any leading and trailing characters (Line 185). Then the mood is returned back to call (Line 187). Lastly, if the integer length of number of items in “list_of_data” is equal to 0, then a null statement through pass is used so it passes over to the rest of the program (Lines 188 – 189).

```
172     @staticmethod
173     def input_mood_to_recommend(list_of_data):
174         """ Input a mood and receive a song suggestion from in the list of dictionary rows
175
176         :param list_of_data: (list) of rows you want to display
177         :return: (string) with mood
178         """
179         if int(len(list_of_data)) > 0:
180             print("Here is the list of available moods to choose from: \n")
181             print(" --- MOODS ---")
182             for row in list_of_data:
183                 print("   ", row["Mood"])
184             print(" ----- \n")
185             mood = str(input(" What is your mood today?: ")).title().strip()
186             print() # Add an extra line for looks
187             return mood
188         elif int(len(list_of_data)) == 0:
189             Pass
```

Inputting to Add New Data into the List

To add a new mood and its suggested song and artist to the Mood Library, first input from the user must be obtained. This, an instruction is printed that states, “Enter a new mood and a song and its artist to match the mood.” (Line 197). Then, three variables are created, “mood”, “song”, and “artist”, that use the `input()` function to firstly obtain data from the user about what the mood is and what is its suggested song and then the artist of the song, respectively, that is then returned as a string through the `str()` function and also uses the `strip()` method to remove leading and trailing characters, such as extra spaces (Lines 199 – 201). An extra line is added afterwards for looks (Line 202). Lastly, the return statement is used to end the function and return the results of “mood”, “song”, and “artist” and use the `title()` method to ensure consistency of the data that is inputted by the user by capitalizing each of the words in the string (Line 203).


```

191     @staticmethod
192     def input_new_mood_song_and_artist():
193         """ Gets mood, song, and artist values to be added to the list
194
195         :return: (string, string, string) with mood, song, and artist
196         """
197         print("Enter a new mood and a song and its artist to match the mood.")
198         print() # Add an extra line for looks
199         mood = str(input(" Enter the mood: ")).strip()
200         song = str(input(" Enter a song: ")).strip()
201         artist = str(input(" Enter the artist: ")).strip()
202         print() # extra line for looks
203         return mood.title(), song.title(), artist.title()

```

Inputting to Remove Existing Data from the List

To remove a mood from the Music Library, first input from the user must be obtained in order to specify which mood they want to have removed. Thus, Line 211 creates a variable, “mood”, that uses the input() function to firstly obtain that information that is then returned as a string through the str() function and also uses the strip() method to remove leading and trailing characters, such as extra spaces. An extra line is added afterwards for looks and then the return statement is used to end the function and return the result of “mood” (Lines 212 – 213).

```

205     @staticmethod
206     def input_mood_to_remove():
207         """ Gets the mood name to be removed from the list
208
209         :return: (string) with mood
210         """
211         mood = str(input("Which mood suggestion would you like to remove?: ")).strip()
212         print() # Add an extra line for looks
213         return mood

```

MAIN BODY OF THE SCRIPT

This final section of the code is similar to the To-Do List from the assignment from Module 06. The major differences between that script and this one are highlighted here where the position of the while loop is brought after the IO.output_menu_tasks() function, showing the current data is made an option instead of appearing every time, a new option to get a song recommendation based on user input for mood is added, as well as a few changes the messages using the print() function are adjusted to better fit the theme of the program. Lastly, an else statement is added to cover for if the user inputs something that is not a number from 1 to 6 based on choices from the menu of options (Lines 255 – 257).

```

218     # Step 1 - When the program starts, Load data from ToDoFile.txt.
219     Processor.read_data_from_file(file_name=strFileName, list_of_data=lstSuggest) # read file data
220
221     # Step 2 - Display a menu of choices to the user

```

```

222     IO.output_menu_tasks() # Shows menu
223     while True:
224         print() # Add an extra line for looks
225         choice_str = IO.input_menu_choice() # Get menu option
226
227         # Step 3 - Process user's menu choice
228         if choice_str.strip() == '1': # Show current data in the list/table
229             IO.output_current_moods_in_list(list_of_data=lstSuggest)
230
231         elif choice_str.strip() == '2': # Get a song recommendation based on your mood
232             mood = IO.input_mood_to_recommend(list_of_data=lstSuggest)
233             lstSuggest = Processor.recommend_data_from_list(mood=mood, list_of_data=lstSuggest)
234
235         elif choice_str.strip() == '3': # Add a new Mood and Song Suggestion
236             mood, song, artist = IO.input_new_mood_song_and_artist()
237             lstSuggest = Processor.add_data_to_list(mood=mood, song=song, artist=artist,
238 list_of_data=lstSuggest)
239             continue # to show the menu
240
241         elif choice_str == '4': # Remove an existing Mood and Song Suggestion
242             mood = IO.input_mood_to_remove()
243             lstSuggest = Processor.remove_data_from_list(mood=mood, list_of_data=lstSuggest)
244             continue # to show the menu
245
246         elif choice_str == '5': # Save Data to File
247             lstSuggest = Processor.save_data_to_file(file_name=strFileName, list_of_data=lstSuggest)
248             print(" Changes Saved to Music Library!\n")
249             continue # to show the menu
250
251         elif choice_str == '6': # Exit Program
252             print(" Exiting Program. Goodbye!")
253             break # by exiting loop
254
255         else:
256             print(" Input not recognized! Please try again.\n")
257             continue

```

Running the Script

By building upon previous assignment's starting scripts as well as the various listing examples provided by Professor Root and in the textbook, *Python® Programming for the Absolute Beginner, Third Edition*, by Michael Dawson, the following completed code that manages a Mood Music Library Playlist is displayed below.

```

1     import pickle
2
3     # Data ----- #
4     strFileName = 'AppData.dat'
5     dict_row = {} # A row of data separated into elements of a dictionary {Mood, Song, Artist}
6     lstSuggest = [] # A list that acts as a 'table' of rows
7     choice_str = "" # Captures the user option selection

```

```

8
9
10 # Processing ----- #
11 class Processor:
12     """ Performs Processing tasks """
13
14     @staticmethod
15     def save_data_to_file(file_name, list_of_data):
16         """ Saves data from a list of dictionary rows to a File
17
18         :param file_name: (string) with name of file:
19         :param list_of_data: (list) you want filled with file data:
20         :return: (list) of dictionary rows
21         """
22         try:
23             file = open(file_name, "wb")
24             pickle.dump(list_of_data, file)
25             file.close()
26         except Exception as e:
27             print("There was an error! Check file permissions.")
28         return list_of_data
29
30     @staticmethod
31     def read_data_from_file(file_name, list_of_data):
32         """ Reads data from a file into a list of dictionary rows
33
34         :param file_name: (string) with name of file:
35         :param list_of_data: (list) you want filled with file data:
36         :return: (list) of dictionary rows
37         """
38         file = open(file_name, "rb")
39         while True:
40             try:
41                 dict_row = pickle.load(file)
42                 for line in dict_row:
43                     mood, song, artist = line["Mood"], line["Song"], line["Artist"]
44                     row = {"Mood": mood.strip(), "Song": song.strip(), "Artist": artist.strip()}
45                     list_of_data.append(row)
46             except EOFError:
47                 Break
48             except Exception as e:
49                 print("There was a general error!")
50                 print(e, e.__doc__, type(e), sep="\n")
51                 Break
52         file.close()
53         return list_of_data
54
55     @staticmethod
56     def recommend_data_from_list(mood, list_of_data):
57         """ Retrieves data from a list of dictionary rows
58
59         :param mood: (string) with name of mood:
60         :param list_of_data: (list) you want filled with file data:

```

```

61         :return: (list) of dictionary rows
62         """
63         for row in list_of_data:
64             if str(mood).lower() == str(row["Mood"]).lower():
65                 print("Since your mood today is ", "", mood, "", ", the song recommended for you is ", "",
66 str(row["Song"]).title(), "", " by ", str(row["Artist"]).title(), sep="", end=".\\n")
67                 print("\\nHope you like it! ☺ > ☺!) / ♥")
68                 Break
69             else:
70                 Pass
71         else:
72             print("Sorry, there is no recommendation for the mood, ", "", mood, "", "! Please try again or select "
73                 "Option 3' to add a new "
74                 "song recommendation for "
75                 "this mood.", sep="")
76         print() # Add an extra line for looks
77         return list_of_data
78
79     @staticmethod
80     def add_data_to_list(mood, song, artist, list_of_data):
81         """ Adds data to a list of dictionary rows
82
83         :param mood: (string) with name of mood:
84         :param song: (string) with name of song:
85         :param artist: (string) with name of artist:
86         :param list_of_data: (list) you want filled with file data:
87         :return: (list) of dictionary rows
88         """
89         new_row = {"Mood": str(mood).strip(), "Song": str(song).strip(), "Artist": str(artist).strip()}
90         list_of_data.append(new_row)
91         return list_of_data
92
93     @staticmethod
94     def remove_data_from_list(mood, list_of_data):
95         """ Removes data from a list of dictionary rows
96
97         :param mood: (string) with name of mood:
98         :param list_of_data: (list) you want filled with file data:
99         :return: (list) of dictionary rows
100         """
101         try:
102             for row in list_of_data:
103                 if mood.lower() == row["Mood"].lower():
104                     list_of_data.remove(row)
105                     print(" Song suggestion for the mood, ", "", mood.title(), "", ", was successfully removed!\\n",
106                         sep="")
107                     Break
108                 else:
109                     Pass
110             else:
111                 print(" Sorry, the song suggestion for mood, ", "", mood.title(), "", ", could not be found. "
112                     "Please try again.\\n", sep="")

```

```

113     except:
114         print("An error has occurred!")
115     return list_of_data
116
117
118 # Presentation (Input/Output) ----- #
119 class IO:
120     """ Performs Input and Output tasks """
121
122     @staticmethod
123     def output_menu_tasks():
124         """ Display a menu of choices to the user
125
126         :return: nothing
127         """
128         print("""
129             Welcome to the Mood Library!
130             This is a music library that recommends a song to you based on your mood.
131             Below is a menu of options from which you can choose from. Enjoy!
132             """)
133         print("""
134             Menu of Options
135             1) View Current Music Library
136             2) Get a Song Suggestion Based on Your Mood
137             3) Add a Song for a Mood
138             4) Remove a Mood and its Song
139             5) Save Changes to Library
140             6) Exit Program""")
141         print() # Add an extra line for looks
142
143     @staticmethod
144     def input_menu_choice():
145         """ Gets the menu choice from a user
146
147         :return: string
148         """
149         choice = str(input("Please choose an option [1 to 6]: ")).strip()
150         print() # Add an extra line for looks
151         return choice
152
153     @staticmethod
154     def output_current_moods_in_list(list_of_data):
155         """ Shows the current song suggestions for moods in the list of dictionaries rows
156
157         :param list_of_data: (list) of rows you want to display
158         :return: nothing
159         """
160         try:
161             if int(len(list_of_data)) > 0:
162                 print("----- YOUR MUSIC LIBRARY -----")
163                 print("MOOD | SONG SUGGESTION")
164                 for row in list_of_data:
165                     print(row["Mood"] + " | " + row["Song"] + " by " + row["Artist"])

```

```

166         print("-----")
167         print() # Add an extra line for looks
168     finally:
169         if int(len(list_of_data)) == 0:
170             print(" No current music in mood library! Please choose 'Option 3' to add songs to the library!\n")
171
172     @staticmethod
173     def input_mood_to_recommend(list_of_data):
174         """ Input a mood and receive a song suggestion from in the list of dictionary rows
175
176         :param list_of_data: (list) of rows you want to display
177         :return: (string) with mood
178         """
179         if int(len(list_of_data)) > 0:
180             print("Here is the list of available moods to choose from: \n")
181             print(" --- MOODS ---")
182             for row in list_of_data:
183                 print("   ", row["Mood"])
184             print(" ----- \n")
185             mood = str(input(" What is your mood today?: ")).title().strip()
186             print() # Add an extra line for looks
187             return mood
188         elif int(len(list_of_data)) == 0:
189             Pass
190
191     @staticmethod
192     def input_new_mood_song_and_artist():
193         """ Gets mood, song, and artist values to be added to the list
194
195         :return: (string, string, string) with mood, song, and artist
196         """
197         print("Enter a new mood and a song and its artist to match the mood.")
198         print() # Add an extra line for looks
199         mood = str(input(" Enter the mood: ")).strip()
200         song = str(input(" Enter a song: ")).strip()
201         artist = str(input(" Enter the artist: ")).strip()
202         print() # extra line for looks
203         return mood.title(), song.title(), artist.title()
204
205     @staticmethod
206     def input_mood_to_remove():
207         """ Gets the mood name to be removed from the list
208
209         :return: (string) with mood
210         """
211         mood = str(input("Which mood suggestion would you like to remove?: ")).strip()
212         print() # Add an extra line for looks
213         return mood
214
215
216 # Main Body of Script ----- #
217
218 # Step 1 - When the program starts, Load data from ToDoFile.txt.

```

```

219 Processor.read_data_from_file(file_name=strFileName, list_of_data=lstSuggest) # read file data
220
221 # Step 2 - Display a menu of choices to the user
222 IO.output_menu_tasks() # Shows menu
223 while True:
224     print() # Add an extra line for looks
225     choice_str = IO.input_menu_choice() # Get menu option
226
227     # Step 3 - Process user's menu choice
228     if choice_str.strip() == '1': # Show current data in the list/table
229         IO.output_current_moods_in_list(list_of_data=lstSuggest)
230
231     elif choice_str.strip() == '2': # Get a song recommendation based on your mood
232         mood = IO.input_mood_to_recommend(list_of_data=lstSuggest)
233         lstSuggest = Processor.recommend_data_from_list(mood=mood, list_of_data=lstSuggest)
234
235     elif choice_str.strip() == '3': # Add a new Mood and Song Suggestion
236         mood, song, artist = IO.input_new_mood_song_and_artist()
237         lstSuggest = Processor.add_data_to_list(mood=mood, song=song, artist=artist,
238 list_of_data=lstSuggest)
239         continue # to show the menu
240
241     elif choice_str == '4': # Remove an existing Mood and Song Suggestion
242         mood = IO.input_mood_to_remove()
243         lstSuggest = Processor.remove_data_from_list(mood=mood, list_of_data=lstSuggest)
244         continue # to show the menu
245
246     elif choice_str == '5': # Save Data to File
247         lstSuggest = Processor.save_data_to_file(file_name=strFileName, list_of_data=lstSuggest)
248         print(" Changes Saved to Music Library!\n")
249         continue # to show the menu
250
251     elif choice_str == '6': # Exit Program
252         print(" Exiting Program. Goodbye!")
253         break # by exiting loop
254
255     else:
256         print(" Input not recognized! Please try again.\n")
257         continue

```

The final result of the script in both PyCharm and the OS Command are shown below in Figures 1 (a and b) and 2, respectively.

```
Run - Assignment07
Run: Assignment07
"/Users/carolinacerda/Documents/_PythonClass/Module07 - Files and Exceptions/Assignment07/venv/bin/python"
"/Users/carolinacerda/Documents/_PythonClass/Module07 - Files and Exceptions/Assignment07/Assignment07.py"

Welcome to the Mood Library!
This is a music library that recommends a song to you based on your mood.
Below is a menu of options from which you can choose from. Enjoy!

Menu of Options
1) View Current Music Library
2) Get a Song Suggestion Based on Your Mood
3) Add a Song for a Mood
4) Remove a Mood and its Song
5) Save Changes to Library
6) Exit Program

Please choose an option [1 to 6]: 1

No current music in mood library! Please choose 'Option 3' to add songs to the library!

Please choose an option [1 to 6]: 3

Enter a new mood and a song and its artist to match the mood.

Enter the mood: happy
Enter a song: replay
Enter the artist: shinee

Please choose an option [1 to 6]: chill

Input not recognized! Please try again.

Please choose an option [1 to 6]: 3

Enter a new mood and a song and its artist to match the mood.

Enter the mood: chill
Enter a song: ghost (feat. amir obé)
Enter the artist: k. roosevelt

Please choose an option [1 to 6]: 3

Enter a new mood and a song and its artist to match the mood.

Enter the mood: sad
Enter a song: us ('us and them' movie theme song)
Enter the artist: eason chan

Please choose an option [1 to 6]: 3

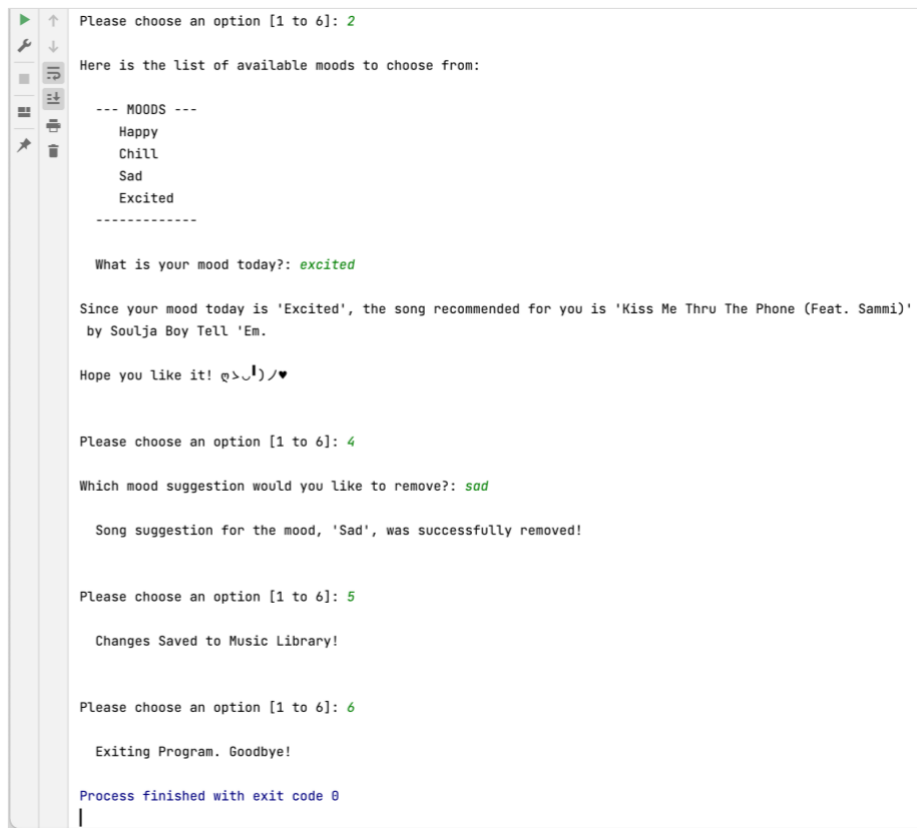
Enter a new mood and a song and its artist to match the mood.

Enter the mood: excited
Enter a song: kiss me thru the phone (feat. sammi)
Enter the artist: soulja boy tell 'em

Please choose an option [1 to 6]: 1

----- YOUR MUSIC LIBRARY -----
MOOD | SONG SUGGESTION
Happy | Replay by Shinee
Chill | Ghost (Feat. Amir Obé) by K. Roosevelt
Sad | Us ('Us And Them' Movie Theme Song) by Eason Chan
Excited | Kiss Me Thru The Phone (Feat. Sammi) by Soulja Boy Tell 'Em
-----
```

Figure 1a. Final result of script in PyCharm (Part 1).



```
Please choose an option [1 to 6]: 2

Here is the list of available moods to choose from:

--- MOODS ---
Happy
Chill
Sad
Excited
-----

What is your mood today?: excited

Since your mood today is 'Excited', the song recommended for you is 'Kiss Me Thru The Phone (Feat. Sammi)'
by Soulja Boy Tell 'Em.

Hope you like it! 🥳!)/♥

Please choose an option [1 to 6]: 4

Which mood suggestion would you like to remove?: sad

Song suggestion for the mood, 'Sad', was successfully removed!

Please choose an option [1 to 6]: 5

Changes Saved to Music Library!

Please choose an option [1 to 6]: 6

Exiting Program. Goodbye!

Process finished with exit code 0
```

Figure 1b. Final result of script in PyCharm (Part 2).

```
carolinacerda — 92x60
Carolinac-MBP:~ carolinacerda$ cd '/Users/carolinacerda/Documents/_PythonClass/Module07 - Files and Exceptions/Assignment07/' && '/usr/local/bin/python3' '/Users/carolinacerda/Documents/_PythonClass/Module07 - Files and Exceptions/Assignment07/Assignment07.py' && echo Exit status: $? && exit 1

Welcome to the Mood Library!
This is a music library that recommends a song to you based on your mood.
Below is a menu of options from which you can choose from. Enjoy!

Menu of Options
1) View Current Music Library
2) Get a Song Suggestion Based on Your Mood
3) Add a Song for a Mood
4) Remove a Mood and its Song
5) Save Changes to Library
6) Exit Program

Please choose an option [1 to 6]: 1

----- YOUR MUSIC LIBRARY -----
MOOD | SONG SUGGESTION
Happy | Replay by Shinee
Chill | Ghost (Feat. Amir Obé) by K. Roosevelt
Excited | Kiss Me Thru The Phone (Feat. Sammi) by Soulja Boy Tell 'Em

Please choose an option [1 to 6]: 2

Here is the list of available moods to choose from:

--- MOODS ---
Happy
Chill
Excited
-----

What is your mood today?: excited

Since your mood today is 'Excited', the song recommended for you is 'Kiss Me Thru The Phone (Feat. Sammi)' by Soulja Boy Tell 'Em.

Hope you like it! 🎵🎶) / ♥

Please choose an option [1 to 6]: 3

Enter a new mood and a song and its artist to match the mood.

Enter the mood: sad
Enter a song: us ('us and them' movie theme song)
Enter the artist: eason chan

Please choose an option [1 to 6]: 4

Which mood suggestion would you like to remove?: excited

Song suggestion for the mood, 'Excited', was successfully removed!

Please choose an option [1 to 6]: 5

Changes Saved to Music Library!

Please choose an option [1 to 6]: 6

Exiting Program. Goodbye!
Exit status: 0
logout

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...27 completed.

[Process completed]
```

Figure 2. Final Result of Script in OS Command.

Finally, a verification of the success of the script is done by confirming that the data inputted by the user has successfully been saved within the binary file, “AppData.dat” (Figure 3). Binary files are not intended to be readable to the human but instead more readily accessible for the computer thus the following result is a rough confirmation that the data was saved correctly.

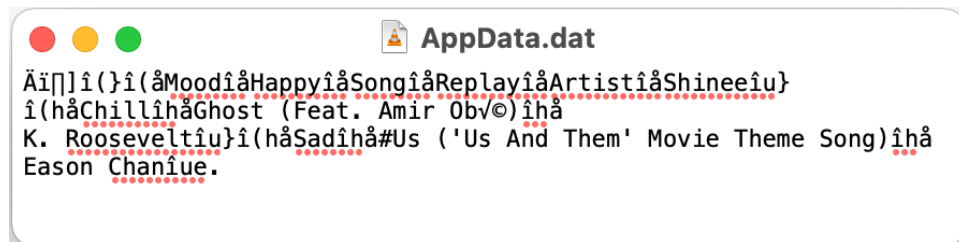


Figure 3. Binary file used in script for data.

Creating a more Advanced GitHub Page

As with Module 06’s assignment, for this module, a GitHub webpage will be created using the same procedure. However, for this week, a more advanced GitHub Page will be created by taking the text, images and links from this document and adding them to the GitHub Page. After creating a new repository for Module 07, “IntroToProg-Python-Mod07”, and creating a new folder, “docs”, and file, “index.md”, and using this as the source to create a GitHub page, the data from this document can now be transferred. Following the instructions on “_Mod07PythonProgramming Notes” as well as from [other online resources](#), the webpage created for this assignment is shown below.

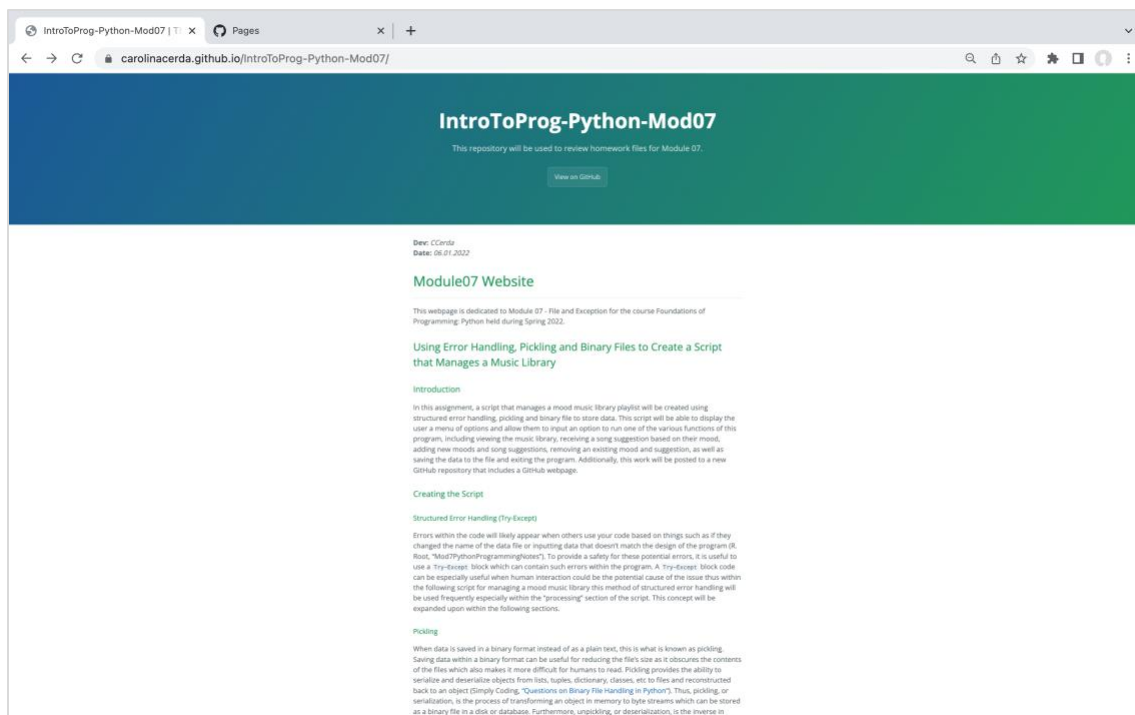


Figure 4. GitHub Webpage for Module 07.

Summary

In this exercise, error handling and pickling were used in order to create a script that manages an interactive Mood Music Library. As no starting script was provided, this investigation was able to promote creativity with the code as well as being able to bring everything we have learned and seen so far together on our own. Additionally, GitHub webpages were expanded upon in further detail in order to practice presenting our work in a more professional and immersive manner for the viewer. The purpose of this investigation was to practice making our own scripts from code we have learned or seen before as well as using error handling and pickling in order to better organize and further improve our scripts as well as to become more familiar with GitHub as a tool for file sharing and storage.