Laurel Warner

11/28/2021

Foundations of Programming

Assignment07

Error Handling and Pickling

# Introduction

For this module, we were introduced to the concept of reading and writing to binary files, which is also known as “pickling”. We also learned about error handling, or how to write code so that if a user enters the wrong thing the whole code doesn’t stop running if you don’t want it to. Also, this module covered the markup language necessary for enhancing GitHub pages.

# Error Handling

Structured error handling is a way to write your code so that if things happen that should not happen then it doesn’t break the code and cause the whole thing to crash. Some examples might be if a user enters the wrong thing for their input, or if a file gets moved or renamed, or if another programmer interacts with the code. You can basically trap the statements that are likely to cause a problem using various methods. One of these is called a “try-except” block. An exception class holds information about the error. Every error that occurs creates an object in an exception class and it holds information about that error. You can write code that shows all of this information, like the type, the arguments, and such. You can derive a new class from an exception class. For example, in a math program, entering input that is not an integer raises a ValueError, but there are other errors that can be raised as part of that, so you can think about the different things and write code to capture them individually. So, you could tell a user that they can’t divide by 0, or a number is too high, or too low, for example. You would create a class derived from the exception class in these instances when there are multiple errors that can be raised in a particular spot in the code. As part of this assignment, I researched Exception Handling in Python. Here are some web pages that I found to be helpful:

[Python Exception Handling Using try, except and finally statement (programiz.com)](https://www.programiz.com/python-programming/exception-handling#:~:text=Python%20Exception%20Handling%20Using%20try%2C%20except%20and%20finally,with%20else%20clause.%20...%206%20Python%20try...finally.%20)[[1]](#footnote-1). I liked the way this page explained the use of the finally clause and how you want to call that at the end of file operations to make sure a file is always closed to ensure that the chance of data corruption is minimized.

[Exception Handling in Python (tutorialsteacher.com)](https://www.tutorialsteacher.com/python/exception-handling-in-python)[[2]](#footnote-2). I liked that the explanations in this were easy for me to understand and I also could follow along with their examples. I was able to mentally follow along and translate what was happening.

Here are some web pages that I didn’t find to be helpful:

[Python - Exceptions Handling (tutorialspoint.com)](https://www.tutorialspoint.com/python/python_exceptions.htm)[[3]](#footnote-3). I didn’t like that this particular tutorial was more advanced than what we have covered up till this point so a lot of what they were covering did not make sense to me. I did like the long list of exceptions that could be raised, though.

[Python Exceptions Handling (With Examples) - Python Guides](https://pythonguides.com/python-exceptions-handling/)[[4]](#footnote-4). This page was not written in plain enough English for me to really follow along. It was like tech folks wrote it for other tech folks and not for people trying to learn who still need things writing in plain language in order to understand what is being talked about.

# Pickling

The differences between a text file and a binary file are that the data in a text file is able to be read easily by a human, but that’s not how the data is actually stored in the computer’s memory. A binary file saves the information in binary code (0 and 1) that the computer can read. This does not require any extra steps to format the data to be saved and retrieved by humans. Saving information in this binary format is called pickling. As part of this assignment, I researched Pickling in Python. Here are some web pages I thought did a good job of explaining the subject:

[Python Pickle | A Comprehensive Guide to Python Pickle (educba.com)](https://www.educba.com/python-pickle/)[[5]](#footnote-5). I liked that the explanations were in pretty plain language. I also did like how they showed how you could pickle things that were a little more complex than basic print statements which many examples tend to be. I like to see examples that cover a wide range of more complex topics that are more likely to come up in actual code.

[What is Pickling in Python? (In-depth Guide) - After nerd](https://www.afternerd.com/blog/python-pickle/)[[6]](#footnote-6). I liked that this page was pretty conversational in tone and was pretty easy to understand. They did go into discussion of other protocols which isn’t necessary for me to know at this point. Also, the author could have put more examples in.

Here are some web pages that I didn’t think did a good job of explaining the subject:

[Python Pickling (tutorialspoint.com)](https://www.tutorialspoint.com/python-pickling)[[7]](#footnote-7). This page was very short and skimpy on details. Also, whoever wrote it must have been working with English as a second language as the grammar and spelling made it hard to focus on the content.

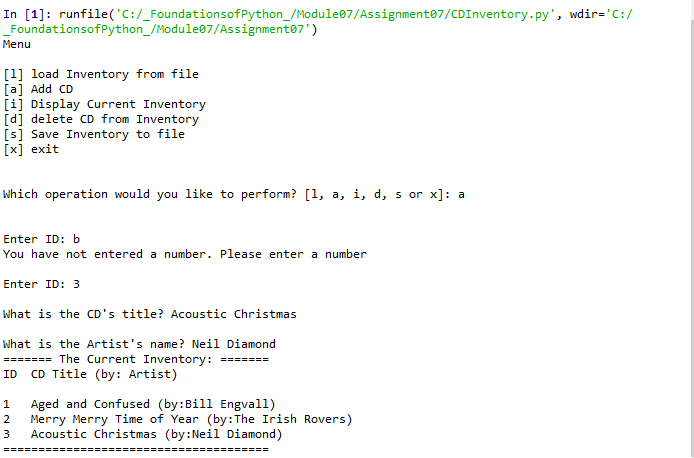
[Understanding Python Pickling with example - GeeksforGeeks](https://www.geeksforgeeks.org/understanding-python-pickling-example/)[[8]](#footnote-8). This was kind of in between. I liked the first example the author used which showed how to pickle a dictionary but the second example was “Pickling without a file” and the author didn’t really go into any explanation at all of what was going on with that example.

# Markdown Language

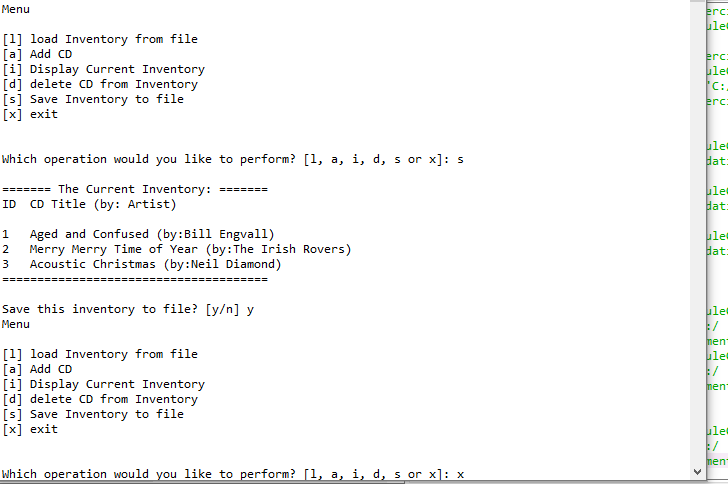
GitHub uses its own kind of markdown language. According to the GitHub document, markdown is a plain text format for writing structured documents. It’s based on conventions for indicating formatting in email and UseNet posts and was developed and released in 2004, so it’s been around a while. Markdown is different from other markup syntaxes because it is more readable. In looking at an example, it does look a bit more like a doc created in a word processing program because indents and things show up there. This might make it a bit harder to type in though.

# Error Handling in the CD Inventory Assignment

For this assignment, I feel like I understood what was being asked, and I understand how to do the error handling, but I got really confused and could not figure out how to actually type it in my code. This is strictly a matter of “what line do I put my ‘while True:’ statement so it will create a loop until we get the correct information?” and “what line should I put “Try” in?” Especially when it comes to the code for deleting a CD from the inventory. Would “try” go before “intRowNR = -1” or “blnCDRemoved = False”, or before “for row in lsTbl:”. And then after what line would I put the Except block? I feel like I KNOW what to do I just end up with so many questions when TRYING to do it. I did work with my tutor but he only had a small amount of time to work with me. Together we were able to do the pickling and unpickling of data, and I did get error handling in the code for Adding a CD but that was it. Here are images of it working in Spyder: (I’m not going to show images of the code doing things that don’t have error handling in them because that’s the same as last week)

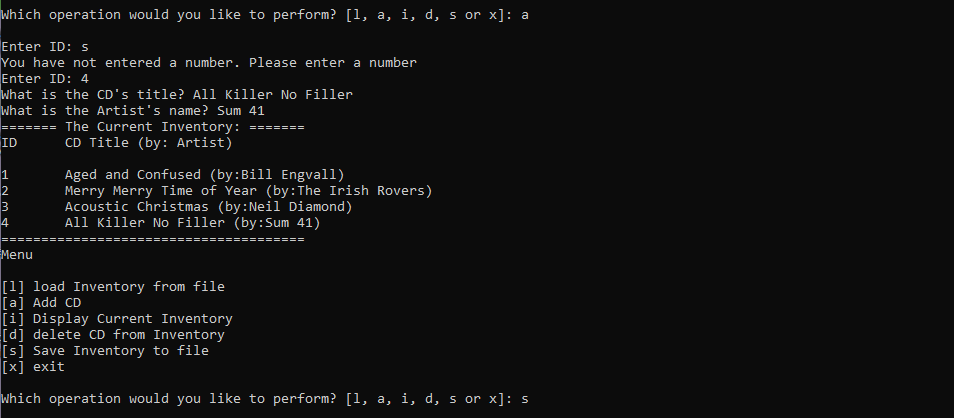


Figure

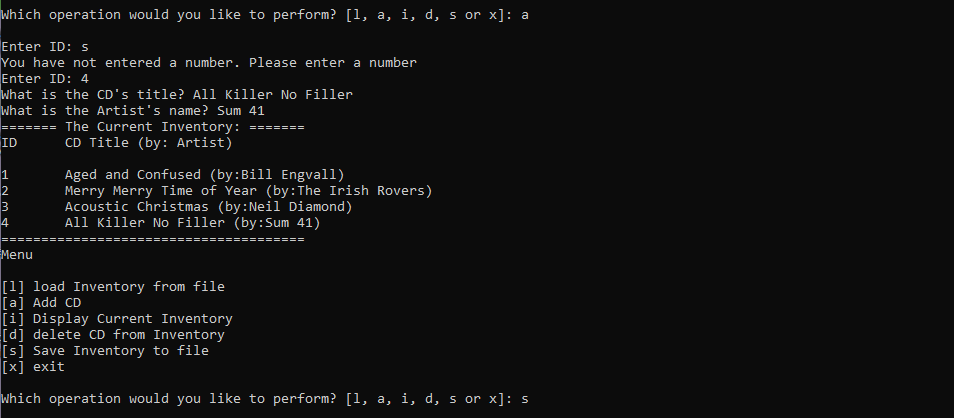


Figure

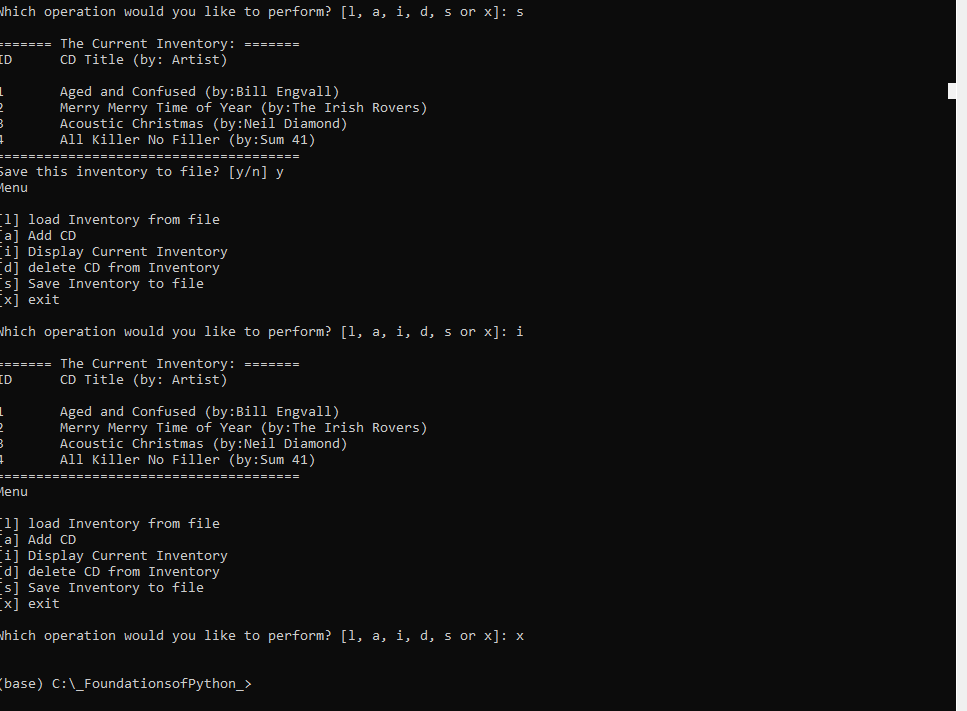
Here are images of the code in the Terminal window. Again, I didn’t capture images where I didn’t have error handling, as that would be the same as last week:



Figure



Figure



Figure

# Summary

In this module I learned the THEORIES of pickling and error handling, but my execution leaves a lot to be desired. I really hope that our next assignment starts out with starter code again because if not I will really be at a disadvantage since I could not complete this assignment adequately. I did add my code to my GitHub Repository at: <https://github.com/lellenn/Assignment_07>.

# Appendix

1. *#------------------------------------------#*
2. *# Title: Assignment07.py*
3. *# Desc: Error Handling and Pickling*
4. *# Change Log: (Who, When, What)*
5. ***# DBiesinger, 2030-Jan-01, Created File***
6. *# LWarner, 2021-Nov-27, Added Code for Error Handling and pickling*
7. *#------------------------------------------#*
9. **import** sys, pickle, pprint
11. *# -- DATA -- #*
12. strChoice = '' *# User input*
13. lstTbl = [] *# list of lists to hold data*
14. dicRow = {} *# list of data row*
15. **strFileName = 'CDInventory.dat' *# data storage file***
16. objFile = None *# file object*

19. *# -- PROCESSING -- #*
20. **class DataProcessor:**
21. """ Functions to add, delete, and save data in the table"""
23. @staticmethod
24. **def** add\_row(dicRow):
25. **"""Function to append a row to the table**
26. Gathers the user input for ID, Title and Artist and appends it to the table in memory.
28. Args:
29. row: the row of ID, Title, and Artist
30. **table: the 2D data structure that holds the data during runtime**
32. Returns:
33. Shows inventory
34. """
35. **dicRow["ID"] = int(dicRow["ID"])**
36. lstTbl.append(dicRow)

39. @staticmethod
40. **def delete\_row(intRowNr, lstTbl):**
41. """ Function to delete a row from the table if the user wants.
43. Args:
44. intRowNr: the ID number for the row to be deleted
45. **lstTbl: the table with the data**
47. Returns:
48. The information showing whether or not the row was successfully deleted.
49. """
50. **intRowNr = -1**
51. blnCDRemoved = False
52. **for** row **in** lstTbl:
53. intRowNr += 1
54. **if** row['ID'] == intIDDel:
55. **del lstTbl[intRowNr]**
56. blnCDRemoved = True
57. **break**
58. **if** blnCDRemoved:
59. **print**('The CD was removed')
60. **else:**
61. **print**('Could not find this CD!')

64. **class** FileProcessor:
65. **"""Processing the data to and from text file"""**
67. @staticmethod
68. **def** read\_file(file\_name):
69. """Function to manage data ingestion from file to a list of dictionaries
71. Reads the data from file identified by file\_name into a 2D table
72. (list of dicts) table one line in the file represents one dictionary row in table.
74. Args:
75. **file\_name (string): name of file used to read the data from**
76. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
78. Returns:
79. None.
80. **"""**
81. **global** lstTbl
82. **try**:
83. objFile = open(file\_name, 'rb')
84. lstTbl = pickle.load(objFile)
85. **objFile.close()**
86. **except** OSError:
87. **print**('File could not be opened!')
88. **except** pickle.UnpicklingError:
89. **print**('File data error')
91. @staticmethod
92. **def** write\_file(file\_name, table): *# this allows the data to be written to the .txt file*
93. """ Function to allow data to be written to the .txt file that has previously been created
95. **Args:**
96. file\_name (string): name of file used to write the data to
97. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
99. Returns:
100. **None.**
101. """
102. **try**:
103. objFile = open(file\_name, 'wb')
104. pickle.dump(table, objFile)
105. **objFile.close()**
106. **except** OSError:
107. **print**('File could not be opened!')
108. **except** pickle.PicklingError:
109. **print**('Error pickling data')
111. *# -- PRESENTATION (Input/Output) -- #*
113. **class** IO:
114. """Handling Input / Output"""
116. @staticmethod
117. **def** print\_menu():
118. """Displays a menu of choices to the user
120. **Args:**
121. None.
123. Returns:
124. None.
125. **"""**
127. **print**('Menu**\n\n**[l] load Inventory from file**\n**[a] Add CD**\n**[i] Display Current Inventory')
128. **print**('[d] delete CD from Inventory**\n**[s] Save Inventory to file**\n**[x] exit**\n**')
130. **@staticmethod**
131. **def** menu\_choice():
132. """Gets user input for menu selection
134. Args:
135. **None.**
137. Returns:
138. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
140. **"""**
141. choice = ' '
142. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
143. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
144. **print**() *# Add extra space for layout*
145. **return choice**
147. @staticmethod
148. **def** show\_inventory(table):
149. """Displays current inventory table

152. Args:
153. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
155. **Returns:**
156. None.
158. """
159. **print**('======= The Current Inventory: =======')
160. **print('ID\tCD Title (by: Artist)\n')**
161. **for** row **in** table:
162. **print**('{}**\t**{} (by:{})'.format(\*row.values()))
163. **print**('======================================')
165. **@staticmethod**
166. **def** enter\_row():
167. """ Gets user input for adding a new CD to the inventory
169. Args:
170. **strID: The ID of the CD to be added**
171. strTitle: the Title of the CD to be added
172. strArtist: the Artist of the CD to be added
174. Returns:
175. **The ID, Title, and Artist info for each CD rendered as a string.**
176. """
177. **while** True:
178. **try**:
179. strID = int(input('Enter ID: ').strip())
180. **break**
181. **except** ValueError:
182. **print**('You have not entered a number. Please enter a number')
184. strTitle = input('What is the CD**\'**s title? ').strip()
185. **strArtist = input('What is the Artist\'s name? ').strip()**
186. **return** { 'ID': strID, 'Title': strTitle, 'Artist': strArtist }

189. *# 1. When program starts, read in the currently saved Inventory*
190. **FileProcessor.read\_file(strFileName)**
192. *# 2. start main loop*
193. **while** True:
194. *# 2.1 Display Menu to user and get choice*
195. **IO.print\_menu()**
196. strChoice = IO.menu\_choice()
198. *# 3. Process menu selection*
199. *# 3.1 process exit first*
200. **if strChoice == 'x':**
201. **break**
202. *# 3.2 process load inventory*
203. **if** strChoice == 'l':
204. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
205. **strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled: ')**
206. **if** strYesNo.lower() == 'yes':
207. **print**('reloading...')
208. FileProcessor.read\_file(strFileName)
209. IO.show\_inventory(lstTbl)
210. **else:**
211. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
212. IO.show\_inventory(lstTbl)
213. **continue** *# start loop back at top.*
214. *# 3.3 process add a CD*
215. **elif strChoice == 'a':**
216. *# 3.3.1 Ask user for new ID, CD Title and Artist*
217. DataProcessor.add\_row(IO.enter\_row())
218. IO.show\_inventory(lstTbl)
219. **continue** *# start loop back at top.*
220. ***# 3.4 process display current inventory***
221. **elif** strChoice == 'i':
222. IO.show\_inventory(lstTbl)
223. **continue** *# start loop back at top.*
224. *# 3.5 process delete a CD*
225. **elif strChoice == 'd':**
226. *# 3.5.1 get Userinput for which CD to delete*
227. *# 3.5.1.1 display Inventory to user*
228. IO.show\_inventory(lstTbl)
229. *# 3.5.1.2 ask user which ID to remove*
230. **intIDDel = int(input('Which ID would you like to delete? ').strip())**
231. *# 3.5.2 search thru table and delete CD*
232. DataProcessor.delete\_row(intIDDel, lstTbl)
233. IO.show\_inventory(lstTbl)
234. **continue** *# start loop back at top.*
235. ***# 3.6 process save inventory to file***
236. **elif** strChoice == 's':
237. *# 3.6.1 Display current inventory and ask user for confirmation to save*
238. IO.show\_inventory(lstTbl)
239. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
240. ***# 3.6.2 Process choice***
241. **if** strYesNo == 'y':
242. *# 3.6.2.1 save data*
243. FileProcessor.write\_file(strFileName, lstTbl)
244. **else**:
245. **input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')**
246. **continue** *# start loop back at top.*
247. *# 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:*
248. **else**:
249. **print**('General Error')

1. <https://www.programiz.com/python-programming/exception-handling#:~:text=Python%20Exception%20Handling%20Using%20try%2C%20except%20and%20finally,with%20else%20clause.%20...%206%20Python%20try...finally.%20> Retrieved: 11/28/2021 [↑](#footnote-ref-1)
2. <https://www.tutorialsteacher.com/python/exception-handling-in-python> Retrieved 11/28/2021 [↑](#footnote-ref-2)
3. <https://www.tutorialspoint.com/python/python_exceptions.htm> Retrieved 11/28/2021 [↑](#footnote-ref-3)
4. <https://pythonguides.com/python-exceptions-handling/> Retrieved 11/28/2021 [↑](#footnote-ref-4)
5. <https://www.educba.com/python-pickle/> Retrieved 11/28/2021 [↑](#footnote-ref-5)
6. <https://www.afternerd.com/blog/python-pickle/> Retrieved 11/28/2021 [↑](#footnote-ref-6)
7. [Python Pickling (tutorialspoint.com)](https://www.tutorialspoint.com/python-pickling) Retrieved 11/28/2021 [↑](#footnote-ref-7)
8. <https://www.geeksforgeeks.org/understanding-python-pickling-example/> Retrieved 11/28/2021 [↑](#footnote-ref-8)