Nick Culbert

2021 Mar 01

IT FDN 110 A Wi 21

Assignment 07

Pickling and Error Handling

# Introduction

For this assignment, I first modified my code from the previous assignment to incorporate Doug’s feedback/changes. Then, I added a with as statement to each instance where I opened the file. Next, I added code to make the program store data in the CDInventory.txt file in binary rather than in a human-readable format. Finally, I added structured error handling in specific instances.

# Research

Before beginning to write code, I first did further research of Python’s pickle module and structured error handling.

For the pickle module, I found a YouTube video called “[Python Pickle](https://youtu.be/6wSFWOleZlc)”[[1]](#footnote-1) (external reference), which I thought was useful and relevant because it used a dictionary as an example. I also found a YouTube video called “[Using the Python pickle Module](https://youtu.be/XzkhtWYYojg)”[[2]](#footnote-2) (external reference), which gives a clear explanation of how pickled info is saved and accessed.

For structured error handling, I found a YouTube video called [“#63 Python Tutorial for Beginners | Exception Handling](https://youtu.be/6SPDvPK38tw)”[[3]](#footnote-3) (external reference), which covered why one would want to use exceptions and included an explanation of the finally block. I also found a YouTube video called [“#25: Python Exception Handling: try…catch…finally | Python for Beginners](https://youtu.be/brICUKrzVR0)”[[4]](#footnote-4) (external reference), which added the consideration of going outside a list’s range.

Additionally, starting with this assignment, I wanted to write my code so that no line was exceptionally long, in order to make it appear cleaner. To do this, I found [a page on Stack Overflow](https://stackoverflow.com/questions/10660435/pythonic-way-to-create-a-long-multi-line-string)[[5]](#footnote-5) (external reference) that explained how to construct long strings over multiple lines. I also found [a page on Real Python](https://realpython.com/python-f-strings/)[[6]](#footnote-6) (external reference) that talked about f-string formatting with multiple lines. I used both these methods for many of the print statements in my program. Figure 1 is an example of this in action.

1. **print**('WARNING: If you continue,'
2. ' all unsaved data will be lost'
3. ' and the Inventory re-loaded from file.')

Figure 1 - A multi-line print statement

Finally, in testing, I kept getting an EOFError when trying to load pickled data from the “CDInventory.txt” file. I found [another page on Stack Overflow](https://stackoverflow.com/questions/24791987/why-do-i-get-pickle-eoferror-ran-out-of-input-reading-an-empty-file) (external reference) that helped clear this up, and as a result I added an error handling line to the read\_file method (Figure 2).

1. **except** EOFError:
2. **print**('Your CD Inventory is empty!')

Figure 2 - Handling the EOFError

# # -- DATA -- #

Before declaring any variables, I imported the pickle module for later use (Figure 3).

1. **import** pickle

Figure 3 - Importing pickle

# # -- PROCESSING -- #

# class DataProcessor

First, I included Doug’s change of making the CD\_delete method return a Boolean in order to move presentation code into the main body and correctly adhere to the Separation of Concerns (Figure 4). I also removed the IO method that I had called from this class per the same concerns.

1. **def** CD\_delete(intIDDel, table):
2. """Deletes a CD based on previous user input
4. Args:
5. intIDDel (integer): the ID of the CD to delete
6. table (list of dicts): holds the CD inventory during runtime
8. Returns:
9. None.
10. """
11. intRowNr = -1
12. blnCDRemoved = False
13. **for** row **in** table:
14. intRowNr += 1
15. **if** row['ID'] == intIDDel:
16. **del** table[intRowNr]
17. blnCDRemoved = True
18. **break**
19. **return** blnCDRemoved

Figure 4 - Adding a Boolean to CD\_delete

In the add\_tbl\_item method, I consolidated the code by changing the first argument to take the intID variable rather than the strID variable (Figure 5).

1. dictRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
2. table.append(dictRow)

Figure 5 - Using an integer as each dictionary's first value

# class FileProcessor

In the read\_file method, I used a with as statement to open the file, which consolidates the code while ensuring that the file closes when the statement has finished executing. Rather than passing the ‘r’ argument into the open() method, I used ‘ab+’ to allow the file to be created if it does not exist while also allowing for reading in binary. This way, I would not have to use structured error handling to account for the FileNotFound error. Finally, I added the seek() method as a precaution to ensure that program always begins reading from the top of the file (Figure 6).

1. **def** read\_file(file\_name, table):
2. """Ingests data from file to a list of dicts
4. Reads the data from file identified by file\_name into a 2D table
5. (list of dicts) each line in the file represents one dictionary
6. row in table.
8. Args:
9. file\_name (string): name of file used to read the data from
10. table (list of dicts): holds the CD inventory during runtime
12. Returns:
13. None.
14. """
15. table.clear()  # clears existing data and allows loading from file
16. **try**:
17. with open(file\_name, 'ab+') as file:
18. file.seek(0)
19. table = pickle.load(file)

Figure 6 - "With As", "ab+", and "seek()"

In addition to adding the error handling for pickle loading from an empty file, I included a general error handling line both as a precaution and to help debug when testing (Figure 7). The latter reason proved to be extremely useful, and in the rest of my code, any time I added structured error handling, I included this general line.

1. **except** Exception as e:
2. **print**('There was an error...\n'
3. f'Error encountered: {e}\n')

Figure 7 - General error handling

I included a finally block that returns the 2D data structure passed in through the method’s second argument. This way, I can use the results of calling this method in the show\_inventory method in the main body (Figure 8).

1. **finally**:
2. **return** table

Figure 8 - Returning the list of dictionaries

With pickling, the write\_file method becomes very concise. While in previous versions of the “CDInventory.py” script, I had used a for loop to iterate through the 2D data structure in order to write easily-readable text to the file, in this version the method simply dumps the 2D data structure, intact, into the code in binary (Figure 9). This works efficiently in this script because the bulk of the script’s functionality depends upon manipulating the 2D data structure.

1. **def** write\_file(file\_name, table):
2. """Writes current inventory data to text file
4. Args:
5. file\_name (string): name of file used to read the data from
6. table (list of dicts): holds the CD inventory during runtime
8. Returns:
9. None.
10. """
11. with open(file\_name, 'ab') as file:
12. pickle.dump(table, file)

Figure 9 - Consolidating with the pickle module

# class IO

The only thing I changed in the print\_menu method was the code’s layout using a multi-line print statement as discussed earlier.

I also did not change the menu\_choice or show\_inventory methods other than to shorten the printed text.

In the get\_new\_CD method, I added a while loop which continues to execute as long as the user has not entered a valid integer for the CD’s ID. Then, I added an except line that specifically addresses the ValueError and displays a custom response which prompts the user to input a valid integer (Figure 10).

1. **def** get\_new\_CD():
2. """Gets user input for adding a new CD
4. Args:
5. None.
7. Returns:
8. intID (integer): CD's ID
9. strTitle (string): CD's title
10. strArtist (string): artist's name
11. """
12. intID = ''
13. **while** type(intID) == str:
14. **try**:
15. intID = int(input('Enter ID: ').strip())
16. **except** ValueError as e:
17. **print**('Fail! You did not enter an integer.\n'
18. f'Error encountered: {e}\n'
19. 'Please enter an integer....')
20. **except** Exception as e:
21. **print**('There was an error...\n'
22. f'Error encountered: {e}\n')
23. strTitle = input('What is the CD\'s title? ').strip()
24. strArtist = input('What is the Artist\'s name? ').strip()
25. **return** intID, strTitle, strArtist

Figure 10 - Using a while loop with error handling

# Main Body

Under the menu choice for deleting a CD, I added the same type of while loop with error handling as in the get\_new\_CD method since the expected error of the user not inputting an integer was the same. Other than that, I made no major functional changes to the main body.

# Running the script

Figures 11 and 12 are screenshots of my script running in Spyder and Terminal, respectively. Figure 13 is a screenshot of the “CDInventory.txt” file with binary text.

Text

Description automatically generated

Figure 11 - Running the script in Spyder

Text

Description automatically generated

Figure 12 - Running the script in Terminal

Text

Description automatically generated

Figure 13 - The resulting binary file

# Summary

After incorporating Doug’s changes from last week, I successfully used with as statements and the pickle module to simplify and shorten my code, and I used structured error handling in key areas to allow for smooth operation of the program.

# Appendix

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # nickculbert, 2021-Feb-21, added functions
7. # nickculbert, 2021-Feb-28, added error handling and pickling
8. #------------------------------------------#
10. # -- DATA -- #
12. **import** pickle
13. strChoice = '' # User input
14. lstTbl = []  # list of lists to hold data
15. dictRow = {}  # list of data row
16. strFileName = 'CDInventory.txt'  # data storage pickle file
17. objFile = None  # file object
19. # -- PROCESSING -- #
21. **class** DataProcessor:
22. """Processing the data in inventory"""
24. @staticmethod
25. **def** add\_tbl\_item(intID, strTitle, strArtist, table):
26. """Adds items to the table lstTbl
28. Args:
29. intID (integer): CD's ID
30. strTitle (string): CD's title
31. strArtist (string): artist's name
32. table (list of dicts): holds the CD inventory during runtime
34. Returns:
35. None.
36. """
37. dictRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
38. table.append(dictRow)
40. @staticmethod
41. **def** CD\_delete(intIDDel, table):
42. """Deletes a CD based on previous user input
44. Args:
45. intIDDel (integer): the ID of the CD to delete
46. table (list of dicts): holds the CD inventory during runtime
48. Returns:
49. blnCDRemoved (boolean): tracks whether or not a CD
50. was actually removed
51. """
52. intRowNr = -1
53. blnCDRemoved = False
54. **for** row **in** table:
55. intRowNr += 1
56. **if** row['ID'] == intIDDel:
57. **del** table[intRowNr]
58. blnCDRemoved = True
59. **break**
60. **return** blnCDRemoved
62. **class** FileProcessor:
63. """Processing the data to and from text file"""
65. @staticmethod
66. **def** read\_file(file\_name, table):
67. """Ingests data from file to a list of dicts
69. Reads the pickled data from file identified by file\_name and assigns
70. that data to the variable table.
72. Args:
73. file\_name (string): name of file used to read the data from
74. table (list of dicts): holds the CD inventory during runtime
76. Returns:
77. table (list of dicts): holds the CD inventory during runtime
78. """
79. table.clear()  # clears existing data and allows loading from file
80. **try**:
81. with open(file\_name, 'ab+') as file:
82. file.seek(0)
83. table = pickle.load(file)
84. **except** EOFError:
85. **print**('Your CD Inventory is empty!')
86. **except** Exception as e:
87. **print**('There was an error...\n'
88. f'Error encountered: {e}\n')
89. **finally**:
90. **return** table
92. @staticmethod
93. **def** write\_file(file\_name, table):
94. """Pickles current inventory data to text file
96. Args:
97. file\_name (string): name of file used to read the data from
98. table (list of dicts): holds the CD inventory during runtime
100. Returns:
101. None.
102. """
103. with open(file\_name, 'ab') as file:
104. pickle.dump(table, file)
106. # -- PRESENTATION (Input/Output) -- #
108. **class** IO:
109. """Handling Input / Output"""
111. @staticmethod
112. **def** print\_menu():
113. """Displays a menu of choices to the user
115. Args:
116. None.
118. Returns:
119. None.
120. """
121. **print**('==== Menu === \n\n'
122. '[l] load Inventory from file \n'
123. '[a] Add CD\n[i] Display Current Inventory \n'
124. '[d] delete CD from Inventory \n'
125. '[s] Save Inventory to file \n'
126. '[x] exit \n')
128. @staticmethod
129. **def** menu\_choice():
130. """Gets user input for menu selection
132. Args:
133. None.
135. Returns:
136. choice (string): a lower case string of the users input
137. out of the choices l, a, i, d, s or x
139. """
140. choice = ' '
141. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
142. choice = input('Please choose an operation: ').lower().strip()
143. **print**()  # extra space for layout
144. **return** choice
146. @staticmethod
147. **def** show\_inventory(table):
148. """Displays current inventory table
150. Args:
151. table (list of dict): holds the CD inventory during runtime
153. Returns:
154. None.
156. """
157. **print**()
158. **print**('======= The Current Inventory: =======')
159. **print**('ID\tCD Title (by: Artist)\n')
160. **for** row **in** table:
161. **print**('{}\t{} (by:{})'.format(\*row.values()))
162. **print**('======================================')
163. **print**()
165. @staticmethod
166. **def** get\_new\_CD():
167. """Gets user input for adding a new CD
169. Args:
170. None.
172. Returns:
173. intID (integer): CD's ID
174. strTitle (string): CD's title
175. strArtist (string): artist's name
176. """
177. intID = ''
178. **while** type(intID) == str:
179. **try**:
180. intID = int(input('Enter ID: ').strip())
181. **except** ValueError as e:
182. **print**('Fail! You did not enter an integer.\n'
183. f'Error encountered: {e}\n'
184. 'Please enter an integer....')
185. **except** Exception as e:
186. **print**('There was an error...\n'
187. f'Error encountered: {e}\n')
188. strTitle = input('What is the CD\'s title? ').strip()
189. strArtist = input('What is the Artist\'s name? ').strip()
190. **return** intID, strTitle, strArtist
192. # 1. When program starts, read in the currently saved Inventory
193. FileProcessor.read\_file(strFileName, lstTbl)
195. # 2. start main loop
196. **while** True:
197. # 2.1 Display Menu to user and get choice
198. IO.print\_menu()
199. strChoice = IO.menu\_choice()
201. # 3. Process menu selection
203. # 3.1 process exit first
204. **if** strChoice == 'x':
205. **break**
207. # 3.2 process load inventory
208. **if** strChoice == 'l':
209. **print**('WARNING: If you continue,'
210. ' all unsaved data will be lost'
211. ' and the Inventory re-loaded from file.')
212. strYesNo = input('Type \'yes\' to continue'
213. ' and reload from file.'
214. ' Type anything else to cancel the reload: ')
215. **if** strYesNo.lower() == 'yes':
216. **print**('\nreloading...\n')
217. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
218. **else**:
219. input('Canceling...Inventory data NOT reloaded.'
220. ' Press [ENTER] to continue to the menu.')
221. IO.show\_inventory(lstTbl)
222. **continue**
224. # 3.3 process add a CD
225. **elif** strChoice == 'a':
226. intID, strTitle, strArtist = IO.get\_new\_CD()
227. DataProcessor.add\_tbl\_item(intID, strTitle, strArtist, lstTbl)
228. IO.show\_inventory(lstTbl)
229. **continue**
231. # 3.4 process display current inventory
232. **elif** strChoice == 'i':
233. IO.show\_inventory(lstTbl)
234. **continue**
236. # 3.5 process delete a CD
237. **elif** strChoice == 'd':
238. IO.show\_inventory(lstTbl)
239. intIDDel = ''
240. **while** type(intIDDel) == str:
241. **try**:
242. intIDDel = int(input('Which ID would'
243. ' you like to delete? ').strip())
244. **except** ValueError as e:
245. **print**('Fail! You did not enter an integer.\n'
246. f'Error encountered: {e}\n'
247. 'Please enter an integer....')
248. **except** Exception as e:
249. **print**('There was an error...\n'
250. f'Error encountered: {e}\n')
251. cd\_removed = DataProcessor.CD\_delete(intIDDel, lstTbl)
252. **if** cd\_removed:
253. **print**('The CD was removed')
254. **else**:
255. **print**('Could not find this CD!')
256. **continue**
258. # 3.6 process save inventory to file
259. **elif** strChoice == 's':
260. IO.show\_inventory(lstTbl)
261. strYesNo = input('Save inventory to file? [y/n] ').strip().lower()
262. **if** strYesNo == 'y':
263. FileProcessor.write\_file(strFileName, lstTbl)
264. **else**:
265. input('The inventory was NOT saved to file.'
266. ' Press [ENTER] to return to the menu.')
267. **continue**
269. # 3.7 catch-all should not be possible, but to be safe:
270. **else**:
271. **print**('General Error')

1. <https://youtu.be/6wSFWOleZlc> retrieved 2021-Mar-01 [↑](#footnote-ref-1)
2. <https://youtu.be/XzkhtWYYojg> retrieved 2021-Mar-01 [↑](#footnote-ref-2)
3. <https://youtu.be/6SPDvPK38tw> retrieved 2021-Mar-01 [↑](#footnote-ref-3)
4. <https://youtu.be/brICUKrzVR0> retrieved 2021-Mar-01 [↑](#footnote-ref-4)
5. <https://stackoverflow.com/questions/10660435/pythonic-way-to-create-a-long-multi-line-string> retrieved 2021-Mar-01 [↑](#footnote-ref-5)
6. <https://realpython.com/python-f-strings/> retrieved 2021-Mar-01 [↑](#footnote-ref-6)