**Name: Srirupa Guha**

**Date**: August 26, 2020

**Course**: IT FDN 110: Introduction to Programming (Python)

**Assignment**: 7

**Title**: Knowledge Document – Working with Text Files, Binary Files and Error Handling

**Introduction** –

This week we learned various text file commands for reading, writing, appending, etc., as well as the readline() function which is a convenient way to read files either one line at a time or several lines. We also learned about binary files and the Pickle class used for dealing with binary files, such as “dumping”, which is a memory dump into a binary file (.dat). Finally, we learned about structured error handling, situations where errors may occur and how to catch them and display information about the errors to the user.

**Text File Operations**

Text files are readable files, usually in the .txt extension. You can easily open up a text file and read the exact contents that are available. The open() function in Python is used for opening up text files, such as:

fileObj = None

fileObj = open(‘thisFile.txt’, ‘r’)

data = fileObj.read() # the data variable holds the contents of the textfile that is read

Note: The ‘r’ here is to indicate that the text file is being opened in “read” format. For writing purposes, the appropriate code would be: fileObj = open(‘thisFile.txt’, ‘w’).

For writing something into a file, one must first open the file in read format and then use code like:

fileObj.write(“This is a piece of text”)

For closing the file: fileObj.close() will do the job. The best coding practice is to embed the read and write functions into some wrapper functions like: read\_file and write\_file.

There is another useful function called the readline() function that reads one line at a time from a file and remembers the location for the next line to read. Usually the syntax is something like this:

fileObj = None

fileObj = open(‘thisFile.txt’, ‘r’)

data = fileObj.readline()

fileObj.close()

Then there is the readlines() function that can read multiple lines of a text file and put it in any defined variable.

We also learned about the With…As manner of file operations such that we don’t have remember to close the file. Here is an example:

with open(‘thisFile.txt’, ‘r’) as fileObj:

data = fileObj.readlines()

So there is no need to issue any file close() command since the file is closed with the coding above after it is read.

Writing into files is easy as well with the With…As statement:

with open(‘thisFile.txt’, ‘w’) as fileObj:

fileObj.write(‘This is a text to write into the file.’)

**Binary Files and Pickling**

Other than text files, there are binary files as well. This kind of file is not readable, since the data is stored in binary format, but yet you can read and write into it. The manner of saving binary files is called pickling, where there’s a memory dump issued into a file and saved in binary format. In order to do this, the pickle class needs to be imported, and when opening binary files, a ‘b’ should be appended to the format, for example:

fileObj = open(‘thisFile.dat’, ‘rb’) # read only format for binary files

or:

fileObj = open(‘thisFile.dat’, ‘wb’) # write format for binary files

You use the pickle.dump() function for “dumping” from memory into a binary file, as in:

fileObj = open(‘thisFile.dat’, ‘wb’)

pickle.dump(dump\_data, fileObj)

pickle.load() function is used for reading from binary files

There is a lot of useful information about the pickling of binary files. One of them is: [Python Pickling](https://www.tutorialspoint.com/python-pickling) [[1]](#footnote-1), which provides the basic syntax of reading/loading and saving/dumping of binary files by the functions of the pickle class. I also found this site by Lokesh Sharma to be very useful, such as: [Pickle in Python by Lokesh Sharma](https://medium.com/@lokeshsharma596/what-is-pickle-in-python-3d9f261498b4) [[2]](#footnote-2)

**Error Handling**

In all applications, error handling is a must, because if the application crashes due to a programming error, all data in the memory is lost. To handle these exceptions, the try… except statement is used around statements-to-perform. If there is any kind of error, such as division by zero, or wrong type of data entered, or file not found, for example, then an exception is “caught” with appropriate messages sent to the user. Structured error handling means writing code in points of error that also sends out all the pertinent information about the error to the user. And example is:

try:

ID = int(input('Enter ID: ').strip())

Title = input('What is the CD\'s title? ').strip()

Artist = input('What is the Artist\'s name? ').strip()

return ID, Title, Artist

except ValueError as e:

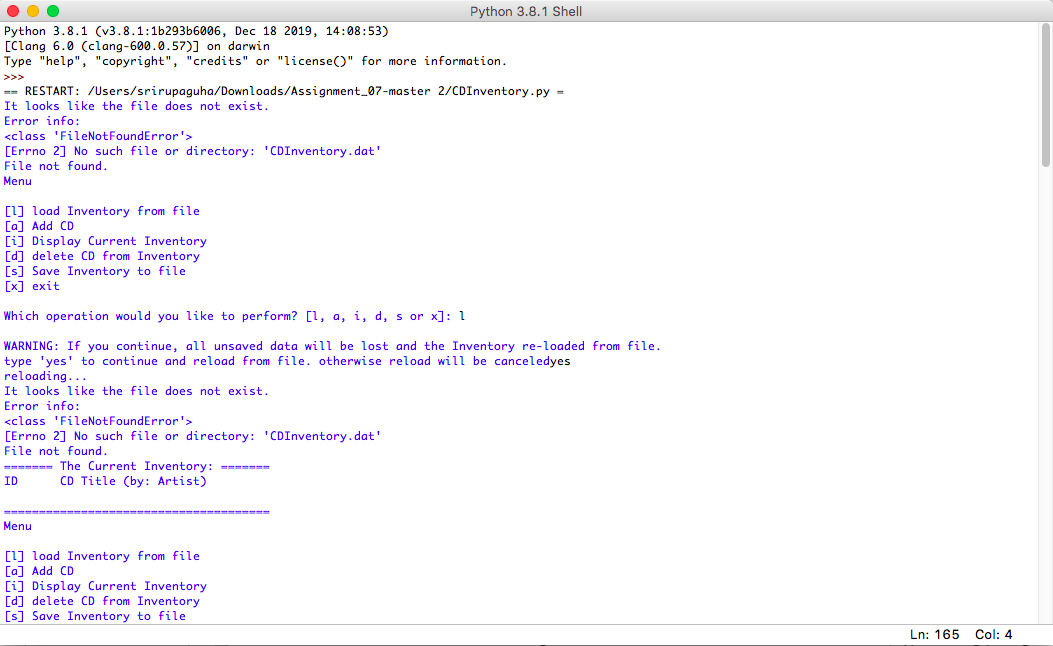
print("Only numbers allowed for ID")

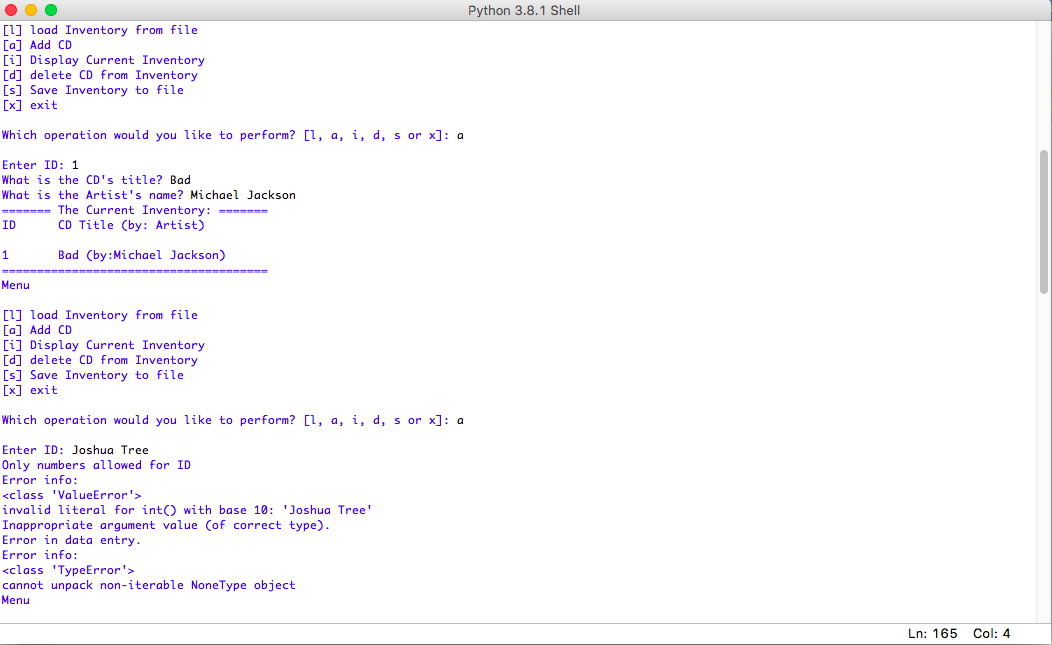
print("Error info: ")

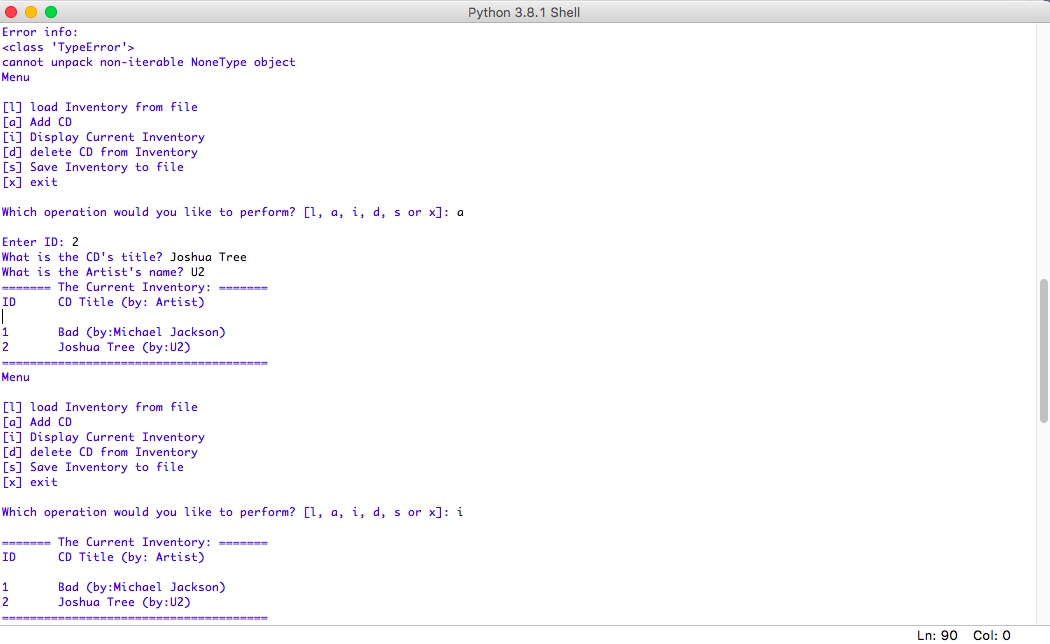
print(type(e),e,e.\_\_doc\_\_, sep="\n")

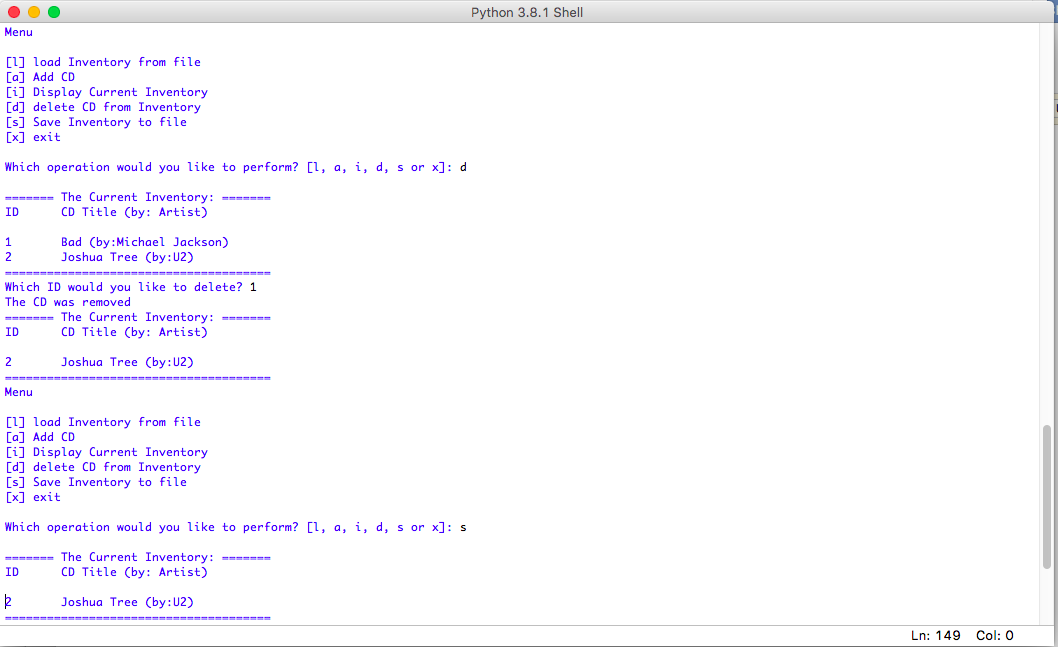
In the example above, only numbers are expected for the ID field, but if the user entered anything else, such as string, for example, the program won’t crash and the messages about the error will be displayed to the user. Some very useful information about error handling are: [Python Try Except](https://www.w3schools.com/python/python_try_except.asp) [[3]](#footnote-3) , which illustrates the syntax of the try…except statement well. Another link goes into more detail about error handling and also illustrates how to raise exceptions: [Errors and Exceptions](Errors%20and%20Exceptions) [[4]](#footnote-4)

Ultimately, I put all this knowledge to test and wrote and executed my Python program:









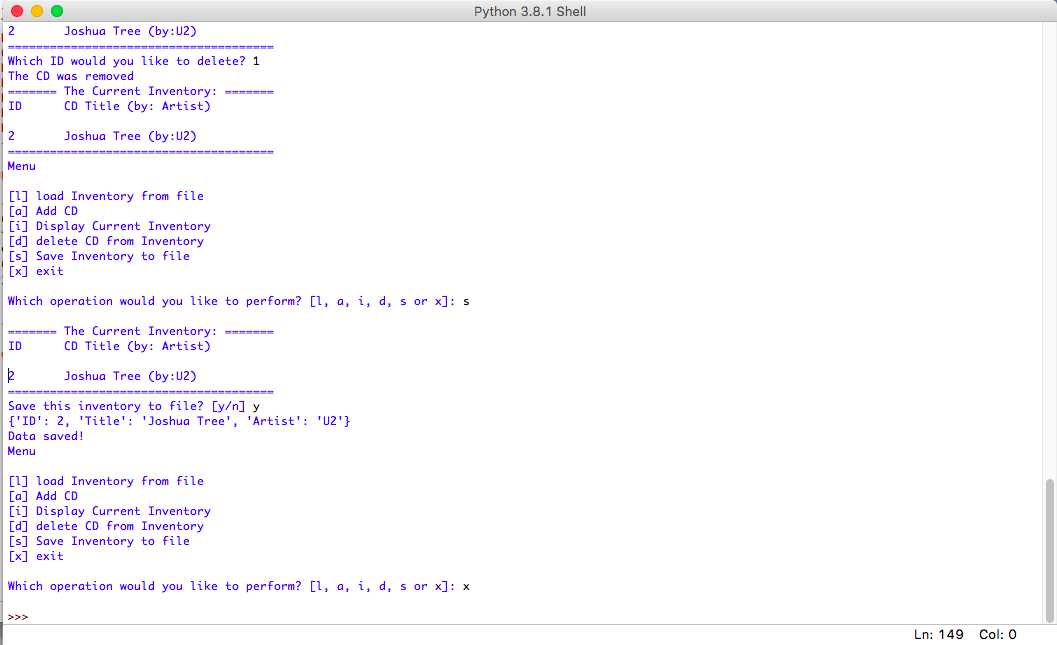


Figure My Python Program

**Summary**

Finally, this week we learned various text file commands for reading, writing, appending, etc., as well as the readline() function which is a convenient way to read files either one line at a time or several lines. We also learned about binary files and the Pickle class used for dealing with binary files, such as “dumping”, which is a memory dump into a binary file (.dat). Finally, we learned about structured error handling, situations where errors may occur and how to catch them and display information about the errors to the user.

**Appendix**

The Python assignment code for this week:

1. #------------------------------------------#
2. # Title: CDInventory.py
3. # Desc: Working with text file operations and error handling.
4. # Change Log: (Who, When, What)
5. # Rupa Guha, 2020-Aug-23, Created File from last week's homework assignment
6. # Rupa Guha, 2020-Aug-25, Modified - implemented suggested changes and corrected errors
7. # Rupa Guha, 2020-Aug-25, Modified - added error handling
8. # Rupa Guha, 2020-Aug-25, Modified - added binary store read/write code
9. # Rupa Guha, 2020-Aug-25, Modified - added the dunder docs to the error handling code
10. #------------------------------------------#
12. **import** pickle
14. # -- DATA -- #
15. strChoice = '' # User input
16. lstTbl = []  # list of lists to hold data
17. dicRow = {}  # list of data row
18. strFileName = 'CDInventory.dat'  # data storage file
19. objFile = None  # file object

22. # -- PROCESSING -- #
24. **class** DataProcessor:
25. # TODone add functions for processing here
26. @staticmethod
27. **def** input\_data\_process(intID, cdTitle, cdArtist, lstTbl):
28. """Function to add user input data to table
30. Reads the data from file identified by file\_name into a 2D table
31. (list of dicts) table one line in the file represents one dictionary row in table.
33. Args:
34. The ID, Title and Artist newly input by the user
36. Returns:
37. None.
38. """
39. dicRow = {'ID': intID, 'Title': cdTitle, 'Artist': cdArtist}
40. lstTbl.append(dicRow)
42. @staticmethod
43. **def** delete\_row(rowId, lstTbl):
44. """Function to delete row from the inventory
46. Args:
47. The ID of the row intended to be deleted
49. Returns:
50. None.
51. """
52. intRowNr = -1
53. blnCDRemoved = False
54. **for** row **in** lstTbl:
55. intRowNr += 1
56. **if** row['ID'] == rowId:
57. **del** lstTbl[intRowNr]
58. blnCDRemoved = True
59. **break**
60. **if** blnCDRemoved:
61. **print**('The CD was removed')
62. **else**:
63. **print**('Could not find this CD!')

66. **class** FileProcessor:
67. """Processing the data to and from text file"""
69. @staticmethod
70. **def** read\_file(file\_name, table):
71. """Function to manage data ingestion from file to a list of dictionaries
73. Reads the data from file identified by file\_name into a 2D table
74. (list of dicts) table one line in the file represents one dictionary row in table.
76. Args:
77. file\_name (string): name of file used to read the data from
78. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
80. Returns:
81. None.
82. """
83. table.clear()  # this clears existing data and allows to load data from file
85. # catching errors like empty file or file not found
86. **try**:
87. objFile = open(file\_name, 'rb')
88. row\_line = []
90. data = pickle.load(objFile)
92. lstData = data.strip().split('\n')
94. **for** item **in** lstData:
95. row\_line = item.strip().split(',')
96. dicRow = {'ID': int(row\_line[0]), 'Title': row\_line[1], 'Artist': row\_line[2]}
97. table.append(dicRow)
99. objFile.close()
101. **except** FileNotFoundError as e:
102. **print**("It looks like the file does not exist.")
103. **print**("Error info: ")
104. **print**(type(e),e,e.\_\_doc\_\_, sep="\n")
106. **except** ValueError as e:
107. **print**("It looks like the file is empty.")
108. **print**("Error info: ")
109. **print**(type(e),e,e.\_\_doc\_\_, sep="\n")

112. **def** save\_file(file\_name, table):
113. """Function to save the text file
115. Writes the data from a 2D table (list of dicts) into a long string and saved into a text file.
117. Args:
118. file\_name (string): name of file used to write the data to
119. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
121. Returns:
122. None.
123. """
124. objFile = open(file\_name, 'wb')
125. new\_line = ""
127. **for** row **in** table:
128. **print**(row)
129. **for** item **in** row.values():
130. new\_line = new\_line + str(item) + ","
131. new\_line = new\_line[:-1] + "\n"
133. pickle.dump(new\_line, objFile)
134. objFile.close()
135. **print**("Data saved!")

138. # -- PRESENTATION (Input/Output) -- #
140. **class** IO:
141. """Handling Input / Output"""
143. @staticmethod
144. **def** print\_menu():
145. """Displays a menu of choices to the user
147. Args:
148. None.
150. Returns:
151. None.
152. """
154. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
155. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
157. @staticmethod
158. **def** menu\_choice():
159. """Gets user input for menu selection
161. Args:
162. None.
164. Returns:
165. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
167. """
168. choice = ' '
169. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
170. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
171. **print**()  # Add extra space for layout
172. **return** choice
174. @staticmethod
175. **def** show\_inventory(table):
176. """Displays current inventory table

179. Args:
180. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
182. Returns:
183. None.
185. """
186. **print**('======= The Current Inventory: =======')
187. **print**('ID\tCD Title (by: Artist)\n')
188. **for** row **in** table:
189. **print**('{}\t{} (by:{})'.format(\*row.values()))
190. **print**('======================================')
192. # TODone add I/O functions as needed
193. @staticmethod
194. **def** ask\_user\_data():
195. """Asks for user data
197. Args: None
198. Returns: The ID, the CD Title and the Artist of the title
199. """
201. # catching errors like entering non-numeric entries
202. **try**:
203. ID = int(input('Enter ID: ').strip())
204. Title = input('What is the CD\'s title? ').strip()
205. Artist = input('What is the Artist\'s name? ').strip()
206. **return** ID, Title, Artist
207. **except** ValueError as e:
208. **print**("Only numbers allowed for ID")
209. **print**("Error info: ")
210. **print**(type(e),e,e.\_\_doc\_\_, sep="\n")

213. # 1. When program starts, read in the currently saved Inventory
214. FileProcessor.read\_file(strFileName, lstTbl)
216. # 2. start main loop
217. **while** True:
218. # 2.1 Display Menu to user and get choice
219. IO.print\_menu()
220. strChoice = IO.menu\_choice()
222. # 3. Process menu selection
224. # 3.1 process exit first
225. **if** strChoice == 'x':
226. **break**
228. # 3.2 process load inventory
229. **if** strChoice == 'l':
230. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
231. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled')
232. **if** strYesNo.lower() == 'yes':
233. **print**('reloading...')
234. FileProcessor.read\_file(strFileName, lstTbl)
235. IO.show\_inventory(lstTbl)
236. **else**:
237. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
238. IO.show\_inventory(lstTbl)
239. **continue**  # start loop back at top.
241. # 3.3 process add a CD
242. **elif** strChoice == 'a':
244. # catching error when erroneous data was not passed from IO.ask\_user\_data()
245. **try**:
246. # 3.3.1 Ask user for new ID, CD Title and Artist
247. # TODone move IO code into function
248. intID, strTitle, stArtist = IO.ask\_user\_data()
250. **except** TypeError as e:
251. **print**("Error in data entry.")
252. **print**("Error info: ")
253. **print**(type(e),e, sep="\n")
254. **continue**
256. # 3.3.2 Add item to the table
257. # TODone move processing code into function
258. DataProcessor.input\_data\_process(intID, strTitle, stArtist, lstTbl)
259. IO.show\_inventory(lstTbl)
260. **continue**  # start loop back at top.
262. # 3.4 process display current inventory
263. **elif** strChoice == 'i':
264. IO.show\_inventory(lstTbl)
265. **continue**  # start loop back at top.
267. # 3.5 process delete a CD
268. **elif** strChoice == 'd':
269. # 3.5.1 get Userinput for which CD to delete
270. # 3.5.1.1 display Inventory to user
271. IO.show\_inventory(lstTbl)
273. # catching error when non-numeric data is entered by user
274. **try**:
275. # 3.5.1.2 ask user which ID to remove
276. intIDDel = int(input('Which ID would you like to delete? ').strip())
277. # 3.5.2 search thru table and delete CD
278. # TODone move processing code into function
279. DataProcessor.delete\_row(intIDDel, lstTbl)
280. **except** ValueError as e:
281. **print**("Only numbers are allowed!")
282. **print**("Error info: ")
283. **print**(type(e),e,e.\_\_doc\_\_, sep="\n")
284. **continue**
286. IO.show\_inventory(lstTbl)
287. **continue**  # start loop back at top.
289. # 3.6 process save inventory to file
290. **elif** strChoice == 's':
291. # 3.6.1 Display current inventory and ask user for confirmation to save
292. IO.show\_inventory(lstTbl)
293. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
294. # 3.6.2 Process choice
295. **if** strYesNo == 'y':
296. # 3.6.2.1 save data
297. # TODone move processing code into function
298. FileProcessor.save\_file(strFileName, lstTbl)
299. **else**:
300. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
301. **continue**  # start loop back at top.
303. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
304. **else**:
305. **print**('General Error')

1. <https://www.tutorialspoint.com/python-pickling> Retrieved August 26, 2020 [↑](#footnote-ref-1)
2. <https://medium.com/@lokeshsharma596/what-is-pickle-in-python-3d9f261498b4> Retrieved August 26, 2020 [↑](#footnote-ref-2)
3. <https://www.w3schools.com/python/python_try_except.asp> Retrieved August 26, 2020 [↑](#footnote-ref-3)
4. <https://docs.python.org/3/tutorial/errors.html> Retrieved August 26, 2020 [↑](#footnote-ref-4)