Kelly Kauffman

August 23, 2020

IT FDN 110: Introduction to Programming: Python

Assignment 07

Module 07:

# Introduction

Module 07’s focus was error handling and pickling. I learned the try/except method to handle possible errors in a program using Python built in error classes. I also learned how pickling/unpickling is similar and different to writing/reading .txt files. Assignment 07 asked for me to use what I learned from Module 07 to modify my Assignment 06 code to handle errors with user inputs and file access operations, and to use pickling to save data in a binary format. In this document I will share the steps I took creating Assignment 07 and the major difficulties I faced.

# Topic 1: Module 07 Learning and Labs

I had a lot more difficulty with the labs in Module 07. In Lab A, the first part I struggled with was loading the two integers from mathIn.txt into my program. Listing #1 in the Appendix shows my first draft of Lab A. When I ran the code for the first time after finishing the TODOs I got an error that said, “ValueError: invalid literal for int() with 10: ‘,4’”. In lines 95 and 96 of the IO.read\_file function (shown in Listing #1 and Figure 1) I thought was telling the function to read the first and third characters of the text file, which were the 8 and 4 I had written in the file beforehand. I was using the example on page 194 of the textbook[[1]](#footnote-1) as a base for the IO.read\_file function.



Figure 1 Line 95 and 96 of IO.read\_file function in Lab A

I initially thought the comma between these two integers that are saved in mathIn.txt was causing a problem when the function, IO.read\_file, was trying to read the two integers. I deleted the comma so I would just have “84”. I then updated line 96 to readline(2) because I thought I would be reading the second character. Looking back on this error and the textbook example days later I realized that line 96 was pulling the second and third characters, “,4”, from the text file. This value was then getting changed to an integer but could not because of the comma. Since I did not realize this at the time I was working on Lab A, I focused on the next error I received, “TypeError: append() takes exactly one argument (2 given)”.

To solve this error I took out the .append in line 98 of Listing #1 and just redefined listofints as [numA, numB]. This solution did the trick and the IO.read\_file function ran without any problems. The rest of the program ran to completion, but I noticed the function IO.write\_file only saved a comma in the text file, mathOut.txt. At this point I was not sure what was amiss with the code, so I added print statements in each of the SimpleMath functions to see if those functions were correctly receiving the integers from mathIn.txt. They were, so something was wrong with the IO.write\_file function.

I tried a couple different solutions to fix IO.write\_file including using a for loop and unpacking the list, lstResults, but I could not get the syntax right. I did some research online and found a solution on stackoverflow.com[[2]](#footnote-2). See Figure 2 for this solution using a for loop.

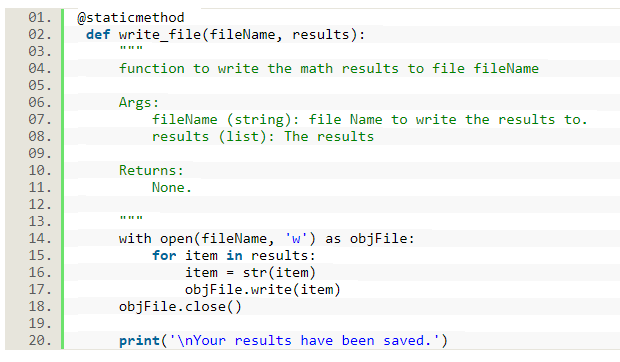


Figure 2 Solution for IO.write\_file function in Lab A

Now, the next item I needed to work on was having my program add commas in between the integers stored in mathOut.txt and take out the comma in between the integers stored on mathIn.txt. After trying the different strategies for formatting that I knew, I settled on the solution in Figure 3. I stripped away the comma in the IO.read\_file function and added a comma during each iteration of the for loop in the IO.write\_file function. I realize that there will be a trailing comma at the end of the list of calculation results that get stored in mathOut.txt. This is something I want to revisit in the future and solve. I also want to come back to this lab and adjust my code to handle the case when more than two integers are stored in mathIn.txt.

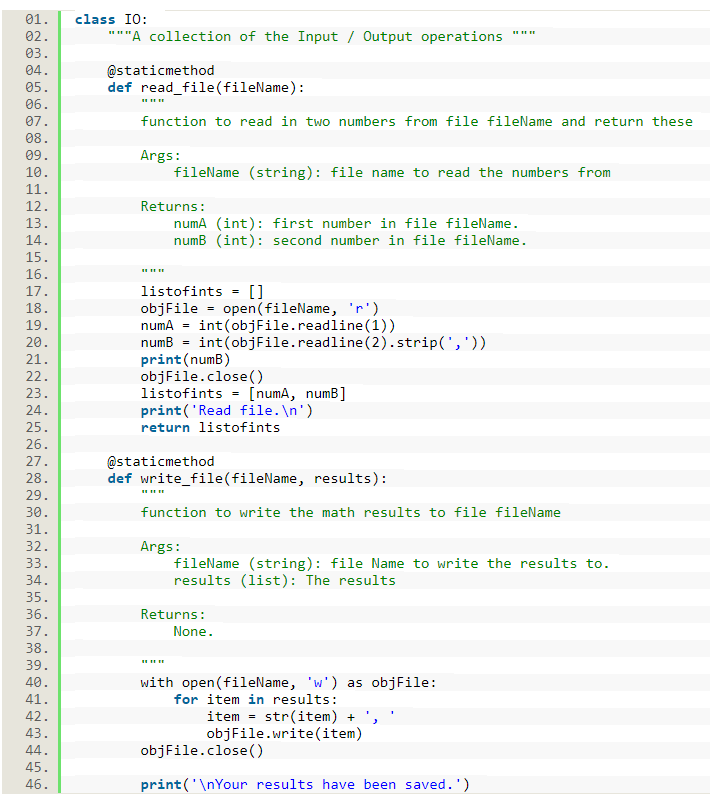


Figure 3 Formatting solution for text files in Lab A

Lab B and C I have yet to complete. I was very confused about how to implement program arguments even after rereading the Module 03 PDF. Since Assignment 07 did not require program arguments I decided to skip Lab B and C for now, so I could focus on Assignment 07. I intend to do further research about program arguments and try to complete Lab B and C later.

To learn more about error handling and pickling I researched some online articles. The first article I found about error handling was on realpython.com[[3]](#footnote-3). I really liked the graphics and examples this webpage had to explain error handling and exceptions. The examples were clear and succinct which made understanding them easier. This article also introduced the topics of assertions and finally, which I do not fully understand yet. The second error handling article I found was on docs.python.org[[4]](#footnote-4). We have looked at several webpages from this webpage during the course of this class. All the webpages from this website are very detailed and informative, which is helpful. I do find that these webpages introduce quite a few new topics that are beyond my understanding right now. In the future when I understand more about Python I think I will find the webpages from docs.python.org to be very useful. The last article I found regarding error handling was from www.pythonforbeginners.com[[5]](#footnote-5). This was not a helpful article. The information provided was vague or unclear. For instance, I was confused what some of the examples in the “Examples of Exception Errors” section were trying to show or explain. I clicked on the “More Reading” links at the bottom of the page and noticed all of them were from 2014 or were for an old version of Python.

The next set of articles I found covered the topic of pickling. The first was from www.geeksforgeeks.org[[6]](#footnote-6). This article was very technical and seemed to be written for more experienced programmers. There were topics that I have never heard of. For example, highest and lowest protocol and pickle class instances. The examples also featured code that seemed a little bit beyond my current understanding. However, from this article I learned that the pickle module is a lot more complex than I originally thought and has a lot of functionality. The next article I found about pickling was from www.datacamp.com[[7]](#footnote-7). I really liked this webpage because it was very clear and gave reasoning behind why pickling can be useful. I learned some new things like the difference between compressing data and pickling it, and that unpickling data that was pickled in a different version of Python can be difficult.

# Topic 2: Further Developing the CD Inventory Program

Before delving fully into Assignment 07, I spent some time implementing the great suggestions and feedback that our TA, Doug, gave me about my Assignment 06 program. In class we were told that a function should be able to operate independently and be imported into another program seamlessly and without a dependence on global variables. I thought the functions I wrote for Assignment 06 met these requirements, but this was not the case. I did not clearly understand what it meant for a function to operate independently until I saw how Doug edited my Assignment 06 program. I was using global variables inside my function instead of passing them into my function as an argument. I transferred Doug’s edits to my Assignment 07 code. Now I believe all my functions could be transferred to another program and operate without any problems.

I was now ready to focus on Assignment 07. The first task I tackled was using error handling to create a text file if one did not already exist. I remember the peer whose Assignment 06 code I reviewed used the try/except method to handle this “FileNotFound” error. See Figure 4 and Listing #2 in the Appendix for this piece of code. I realized as I was writing this try/except code out that in Assignment 06 this “FileNotFound” error is occurring because at the start of the program we are asking it to read a file, which does not create a new file like writing to or appending a file would. I was wondering for some time why there was this difference with Assignment 06 compared with the previous assignments when the text file was created automatically when the program ran for the first time.

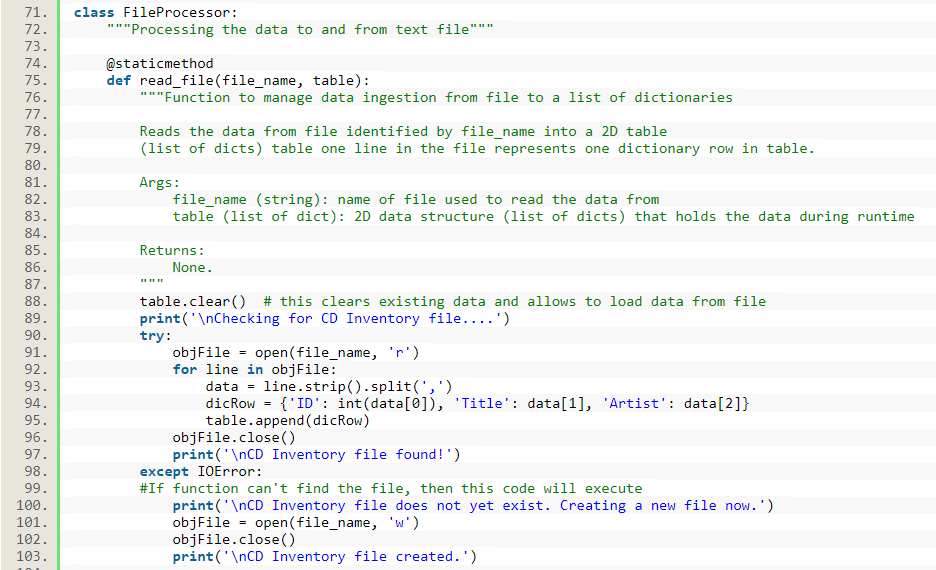


Figure 4 Error handling in Assignment 07 when the text file does not exist

The next TODO I worked on was coding the error handling for the IO.user\_inputs function. I knew I wanted to use a while loop to prompt the user to try to enter the CD ID again if they inputted something other than an integer. I found I needed to go back to the Module 03 PDF and my Assignment 03 code to help me remember how to write a while loop. I also knew when coding this error handling of the CD ID input, that I would need to move the step where the input is turned into an integer (int) from the DataProcessor.create\_table function to the IO.user\_inputs function. This was because if the user did enter a non-integer then I could have them try again right away instead of after they had already inputted the CD title and artist. I did not experience any difficulties using try/except for this user input and the one in the function, DataProcessor.delete\_data besides a couple simple syntax errors.

The next step was to replace all the reading and writing of text files to pickling and unpickling of .dat files. When using pickling for the first time I just focused on being able to create a new .dat file, load previous data, and save data. I left in the formatting that was necessary for saving and loading to/from a text file for the time being. After reading the Module 07 PDF, I remembered that I do not need to format my data because in the process of pickling the data is saved as binary data. I eventually took the formatting steps out. See Figure 5 for how I was pickling and unpickling at this stage in developing Assignment 07. Please note that the code in Figure 5 replaces lines 71-125 from my first Assignment 07 draft in Listing #2 in the Appendix.

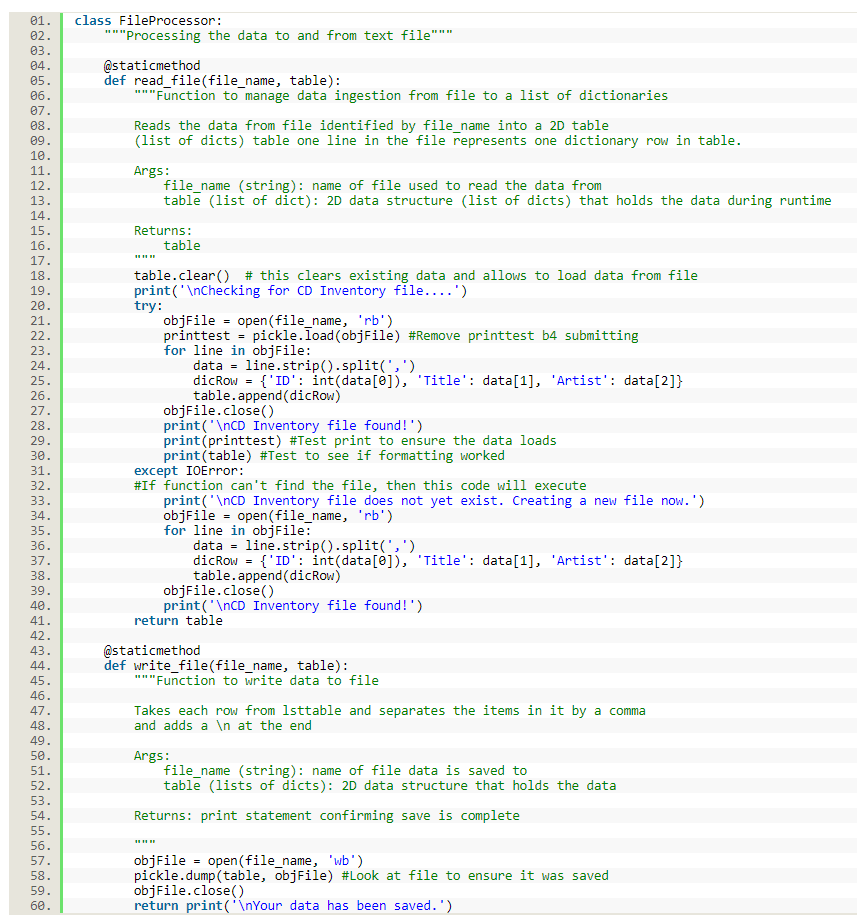


Figure 5 First attempt at pickling/unpickling in Assignment 07

At first, I thought the pickling/unpickling was working, but eventually realized, with the help of test print statements, that my data was not loading into the program’s memory. The solution was to remove the for loop and instead append the variable, printtest, directly to the variable, table (see Figure 6).

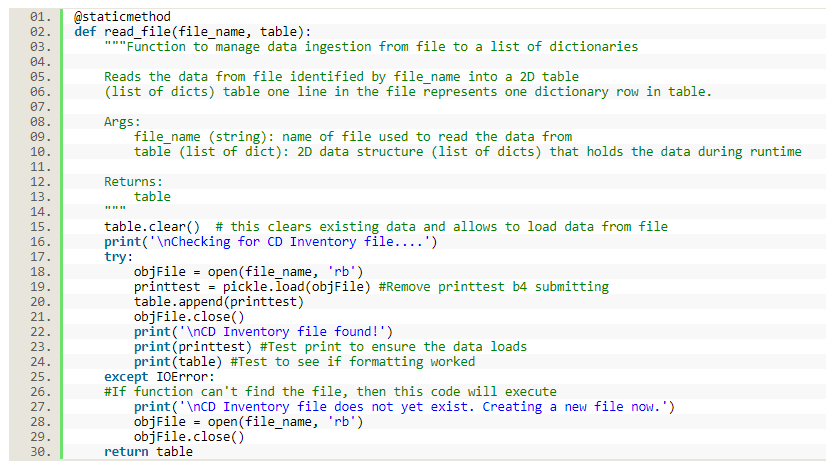


Figure 6 Solution to load previously saved CD data into the program's memory

Now, when I loaded the data from CDInventory.dat, the data was not showing up in the current CD Inventory. See Figure 7 for a visual of what was happening. I discovered my code was appending the list of dictionaries that was being loaded from the .dat file to another list, table. So, I had a list of a list of dictionaries. You can see the printout of this data structure in Figure 7. My solution, shown in Figure 8, was to remove the code relating to appending the table and just make pickle.load(objFile) the value of table.

A screenshot of a cell phone

Description automatically generated

Figure 7 CD data was not showing up in the current CD inventory

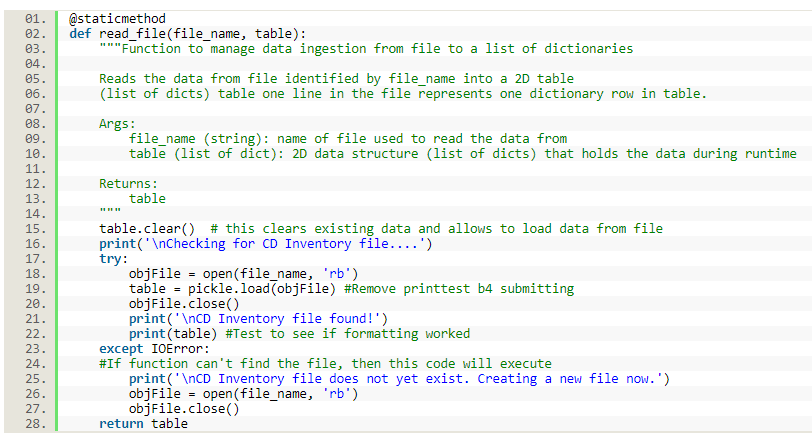


Figure 8 Solution to have the CD data show up in current CD inventory

# Topic 3: Final Products

Here is the link to the final code for Assignment 07 on Github:

<https://github.com/pythongal6295/Assignment_07.git>

Below, Figure 9 and 10 show my code working in Spyder. Figure 11 shows my code working in the Anaconda Prompt.

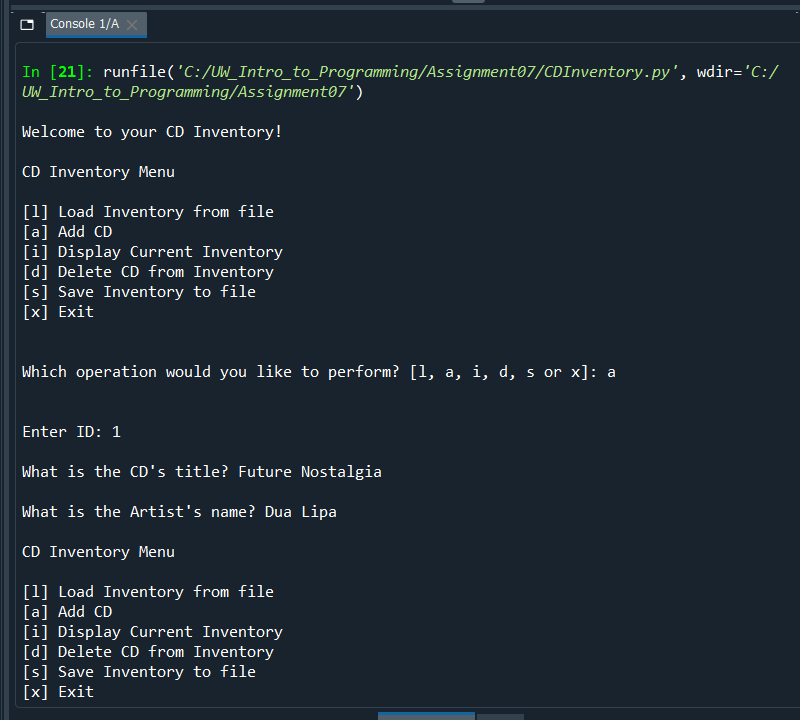


Figure 9 Assignment 07 working in Spyder, part one

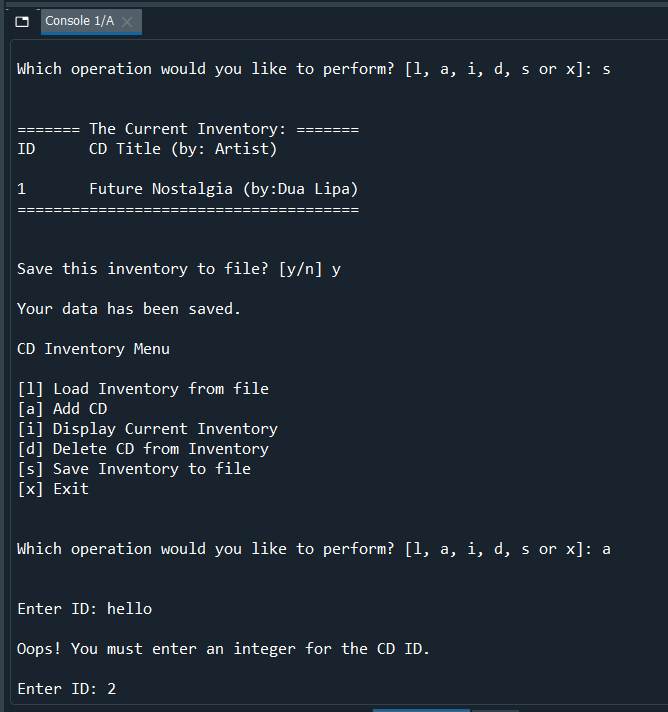


Figure 10 Assignment 07 working in Spyder, part two

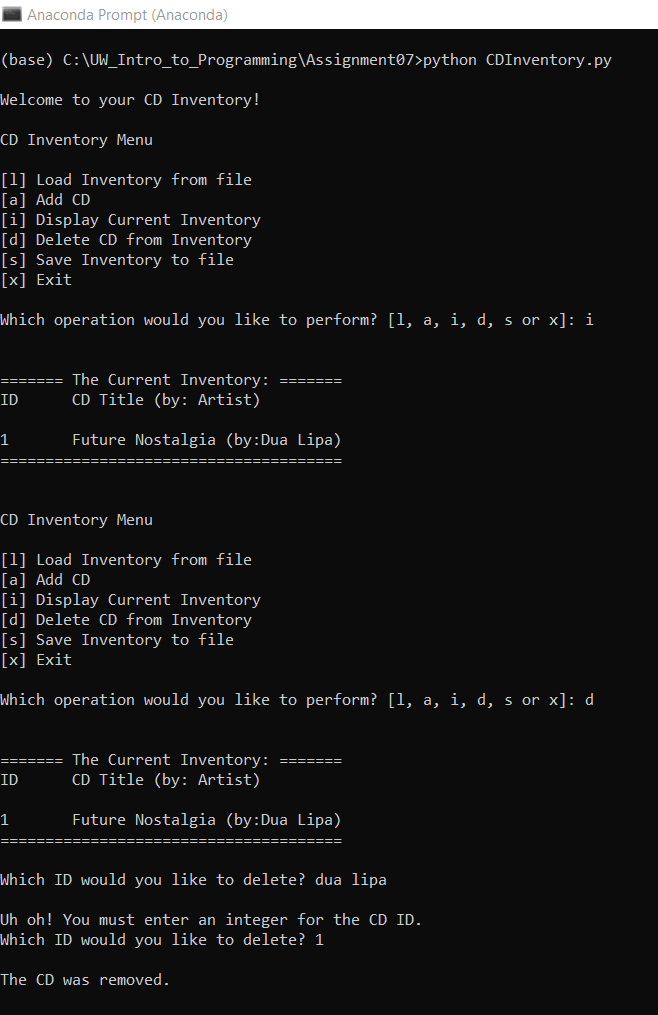


Figure 11 Assignment 07 running in the Anaconda Prompt

# Summary

Module 07 covered error handling, exception classes, and pickling. I learned one method for catching and managing errors is to use try/except. I did not have a lot of trouble using the try/except method in this module. I also understood the logic behind pickling but struggled with a few syntax errors when applying pickling to Assignment 07. The task for Assignment 07 was to use my code from Assignment 06 and add error handling to the functions that deal with reading the external file and user inputs. I was also asked to save all data into binary format using pickling.

# Appendix

Listing #1: First Draft of Lab A

1. #------------------------------------------#
2. # Title: LAB06\_C.py
3. # Desc: simple demonstrator for classes
4. # Change Log: (Who, When, What)
5. # KKauffman, Created File
6. #------------------------------------------#
8. # -- DATA -- #
9. strFileInput = 'mathIn.txt'
10. strFileOutput = 'mathOut.txt'
12. # -- PROCESSING -- #
13. **class** SimpleMath:
14. """A collection of simple math processing functions """
16. @staticmethod
17. **def** get\_sum(val1 = 0.0, val2 = 0.0):
18. """Function for adding two values

21. Args:
22. val1: the first number to add
23. val2: the second number to add

26. Returns:
27. A float corresponding to the sum of val1 and val2
28. """
29. **return** float(val1 + val2)
31. @staticmethod
32. **def** get\_diffference(val1 = 0.0, val2 = 0.0):
33. """Function for subtracting two values

36. Args:
37. val1: the number to subtract from
38. val2: the number to subtract

41. Returns:
42. A float corresponding to the difference of val1 and val2
43. """
44. **return** float(val1 - val2)
46. @staticmethod
47. **def** get\_product(val1 = 0.0, val2 = 0.0):
48. """Function for multiplying two values

51. Args:
52. val1: the first number to multiply
53. val2: the second number to multiply

56. Returns:
57. A float corresponding to the product of val1 and val2
58. """
59. **return** float(val1 \* val2)
61. @staticmethod
62. **def** get\_quotient(val1 = 0.0, val2 = 0.0):
63. """Function for dividing two values

66. Args:
67. val1: the number to divide
68. val2: the number to divide by

71. Returns:
72. A float corresponding to the quotient of val1 and val2
73. """
74. **return** float(val1 / val2)

77. **class** IO:
78. """A collection of the Input / Output operations """
80. @staticmethod
81. **def** read\_file(fileName):
82. """
83. function to read in two numbers from file fileName and return these
85. Args:
86. fileName (string): file name to read the numbers from
88. Returns:
89. numA (int): first number in file fileName.
90. numB (int): second number in file fileName.
92. """
93. listofints = []
94. objFile = open(fileName, 'r')
95. numA = int(objFile.readline(1))
96. numB = int(objFile.readline(3))
97. objFile.close()
98. listofints.append(numA, numB)
100. **return** listofints
102. @staticmethod
103. **def** write\_file(fileName, results):
104. """
105. function to write the math results to file fileName
107. Args:
108. fileName (string): file Name to write the results to.
109. results (list): The results
111. Returns:
112. None.
114. """
115. results = results, sep= ', '
116. objFile = open(fileName, 'w')
117. objFile.writelines(results)
118. objFile.close()
120. **print**('\nYour results have been saved.')
122. # -- PRESENTATION (Input/Output) -- #
123. **print**('Basic Math script. Calculating the Sum, Difference, Product and Quotient of two numbers.')
124. intNumA, intNumB = IO.read\_file(strFileInput)
125. lstResults = []
126. lstResults.append(SimpleMath.get\_sum(intNumA, intNumB))
127. lstResults.append(SimpleMath.get\_diffference(intNumA, intNumB))
128. lstResults.append(SimpleMath.get\_product(intNumA, intNumB))
129. lstResults.append(SimpleMath.get\_quotient(intNumA, intNumB))
130. IO.write\_file(strFileOutput, lstResults)

Listing #2: First Draft of Assignment 07 that includes error handling for user input errors and when a file is not found

1. #------------------------------------------#
2. # Title: Assignment06\_Starter.py
3. # Desc: Working with classes and functions.
4. # Change Log: (Who, When, What)
5. #Assignment 06
6. # KKauffman, 17 Aug 2020, Created File, wrote code for all TODOs,
7. #debugging 'a' menu functions
8. #KKauffman, 18 Aug 2020, fixed 'a' menu functions, formatting edits,
9. #KKauffman, 19 Aug 2020 formatting edits, added details to docstrings
10. #Assignment 07
11. #KKauffman, 24 Aug 2020 edited based on DKlos' recommendations
12. #(e.g. removed dependance on global variables), added error trapping for reading file and user inputs
13. #------------------------------------------#
15. # -- DATA -- #
16. strChoice = '' # User input
17. lstTbl = []  # list of dicts to hold data
18. dicRow = {}  # dict of data row
19. strFileName = 'CDInventory.txt'  # data storage file
20. objFile = None  # file object

23. # -- PROCESSING -- #
24. **class** DataProcessor:
25. """Collect and process user inputs"""
27. @staticmethod
28. **def** create\_table(strID, strTitle, strArtist, table):
29. """Function to take list of user inputs and put them in a dictionary (dicRow)
31. Arg:
32. table/list to store user inputs
33. Taken from unpacked tuple returned by IO.user\_inputs()
34. strID: User's inputted CD ID
35. strTitle: User's inputted CD title
36. strArtist: User's inputted CD Artist
38. Return:
39. None
41. """
43. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
44. table.append(dicRow)
46. @staticmethod
47. **def** delete\_data(id\_to\_remove, table):
48. """Function to delete CD data from memory
50. Arg:
51. table/list that data will be deleted from
52. CD ID that the user wants deleted
54. Return:
55. None
57. """
58. intRowNr = -1
59. blnCDRemoved = False
60. **for** row **in** table:
61. intRowNr += 1
62. **if** row['ID'] == id\_to\_remove:
63. **del** table[intRowNr]
64. blnCDRemoved = True
65. **break**
66. **if** blnCDRemoved:
67. **print**('\nThe CD was removed.\n')
68. **else**:
69. **print**('\nCould not find this CD!\n')
71. **class** FileProcessor:
72. """Processing the data to and from text file"""
74. @staticmethod
75. **def** read\_file(file\_name, table):
76. """Function to manage data ingestion from file to a list of dictionaries
78. Reads the data from file identified by file\_name into a 2D table
79. (list of dicts) table one line in the file represents one dictionary row in table.
81. Args:
82. file\_name (string): name of file used to read the data from
83. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
85. Returns:
86. None.
87. """
88. table.clear()  # this clears existing data and allows to load data from file
89. **print**('\nChecking for CD Inventory file....')
90. **try**:
91. objFile = open(file\_name, 'r')
92. **for** line **in** objFile:
93. data = line.strip().split(',')
94. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
95. table.append(dicRow)
96. objFile.close()
97. **print**('\nCD Inventory file found!')
98. **except** IOError:
99. #If function can't find the file, then this code will execute
100. **print**('\nCD Inventory file does not yet exist. Creating a new file now.')
101. objFile = open(file\_name, 'w')
102. objFile.close()
103. **print**('\nCD Inventory file created.')
105. @staticmethod
106. **def** write\_file(file\_name, table):
107. """Function to write data to file
109. Takes each row from lsttable and separates the items in it by a comma
110. and adds a \n at the end
112. Args:
113. file\_name (string): name of file data is saved to
114. table (lists of dicts): 2D data structure that holds the data
116. Returns: print statement confirming save is complete
118. """
119. objFile = open(file\_name, 'w')
120. **for** row **in** table:
121. lstValues = list(row.values())
122. lstValues[0] = str(lstValues[0])
123. objFile.write(','.join(lstValues) + '\n')
124. objFile.close()
125. **return** **print**('\nYour data has been saved.')
127. # -- PRESENTATION (Input/Output) -- #
129. **class** IO:
130. """Handling Input / Output"""
132. @staticmethod
133. **def** print\_menu():
134. """Displays a menu of choices to the user
136. Args:
137. None.
139. Returns:
140. None.
141. """
143. **print**('\nCD Inventory Menu\n\n[l] Load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
144. **print**('[d] Delete CD from Inventory\n[s] Save Inventory to file\n[x] Exit\n')
146. @staticmethod
147. **def** menu\_choice():
148. """Gets user input for menu selection
150. Args:
151. None.
153. Returns:
154. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
156. """
157. choice = ' '
158. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
159. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
160. **print**()  # Add extra space for layout
161. **return** choice
163. @staticmethod
164. **def** show\_inventory(table):
165. """Displays current inventory table

168. Args:
169. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
171. Returns:
172. None.
174. """
175. **print**()
176. **print**('======= The Current Inventory: =======')
177. **print**('ID\tCD Title (by: Artist)\n')
178. **for** row **in** table:
179. **print**('{}\t{} (by:{})'.format(\*row.values()))
180. **print**('======================================')
181. **print**()
183. @staticmethod
184. **def** user\_inputs():
185. """Function to gather the user's inputs for CD ID, CD Title, and CD Artist
187. Arg: none
189. Return: a tuple of the three user inputs (entryID, entryTitle, entryArtist)
191. """
192. #Will catch if the user enters an noninteger for the ID. Loop will continue to prompt them.
193. entryID = ''
194. **while** True:
195. **try**:
196. entryID = int(input('Enter ID: ').strip())
197. **break**
198. **except** ValueError:
199. **print**('\nOops! You must enter an integer for the CD ID.')
200. **continue**
202. entryTitle = input('What is the CD\'s title? ').strip()
203. entryArtist = input('What is the Artist\'s name? ').strip()
204. **return** (entryID, entryTitle, entryArtist)

207. # 1. When program starts, read in the currently saved Inventory
209. **print**('\nWelcome to your CD Inventory!')
211. #Need to ensure CDInvetory.txt is created before running this program
212. FileProcessor.read\_file(strFileName, lstTbl)

215. # 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\n')
232. **if** strYesNo.lower() == 'yes':
233. **print**('\nreloading...')
234. FileProcessor.read\_file(strFileName, lstTbl)
235. IO.show\_inventory(lstTbl)
236. **else**:
237. input('\ncanceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.\n')
238. IO.show\_inventory(lstTbl)
239. **continue**  # start loop back at top.
241. # 3.3 process add a CD
242. **elif** strChoice == 'a':
244. # 3.3.1 Ask user for new ID, CD Title and Artist
245. strID, strTitle, strArtist = IO.user\_inputs() #Assigned return to variables and unpacked this tuple
247. # 3.3.2 Add item to the table
248. DataProcessor.create\_table(strID, strTitle, strArtist, lstTbl) #Arguments are unpacked tuple from IO.user\_inputs()
249. **continue**  # start loop back at top.
251. # 3.4 process display current inventory
252. **elif** strChoice == 'i':
253. IO.show\_inventory(lstTbl)
254. **continue**  # start loop back at top.
256. # 3.5 process delete a CD
257. **elif** strChoice == 'd':
259. # 3.5.1 get Userinput for which CD to delete
260. # 3.5.1.1 display Inventory to user
261. IO.show\_inventory(lstTbl)
263. # 3.5.1.2 ask user which ID to remove
264. intIDDel = int(input('Which ID would you like to delete? ').strip())
266. # 3.5.2 search thru table and delete CD
267. DataProcessor.delete\_data(intIDDel, lstTbl)
268. IO.show\_inventory(lstTbl)
269. **continue**  # start loop back at top.
271. # 3.6 process save inventory to file
272. **elif** strChoice == 's':
274. # 3.6.1 Display current inventory and ask user for confirmation to save
275. IO.show\_inventory(lstTbl)
276. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
278. # 3.6.2 Process choice
279. **if** strYesNo == 'y':
281. # 3.6.2.1 save data
282. FileProcessor.write\_file(strFileName, lstTbl)
284. **else**:
285. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
286. **continue**  # start loop back at top.
288. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
289. **else**:
290. **print**('General Error')

1. (Dawson, 2010) Dawson, M. (2010). *Python programming for the absolute beginner* (3rd ed.). Australia: Course Technology PTR, a part of Cengage Learning.’ [↑](#footnote-ref-1)
2. <https://stackoverflow.com/questions/899103/writing-a-list-to-a-file-with-python> (Accessed August 23) [↑](#footnote-ref-2)
3. <https://realpython.com/python-exceptions/> (Accessed August 23) [↑](#footnote-ref-3)
4. <https://docs.python.org/3/tutorial/errors.html> (Accessed August 24) [↑](#footnote-ref-4)
5. <https://www.pythonforbeginners.com/error-handling/exception-handling-in-python> (Accessed August 24) [↑](#footnote-ref-5)
6. <https://www.geeksforgeeks.org/pickle-python-object-serialization/> (Accessed August 24) [↑](#footnote-ref-6)
7. <https://www.datacamp.com/community/tutorials/pickle-python-tutorial> (Accessed August 24) [↑](#footnote-ref-7)