Brent Kieszling

8/26/2020

IT FDN 110: Introduction to Programming (Python)

Module 7 Learning Document

# Introduction

In this paper I will cover the standout points of Module 7. Some of the topics to expect are:binary files, the ‘pickle’ function, structured error handling, and custom error handling. Additionally, I will discuss notes on the development of the program I am submitting for Assignment 7.

# Lessons

Previously, we learned how to save information to a text file. Binary files are another way to store program information. The benefit to using binary files is that you can store whole objects instead of converting information to text. Storing objects requires the use of the function ‘pickle’. Figure one below shows an example of saving a list to a data file and then reading the list back in.

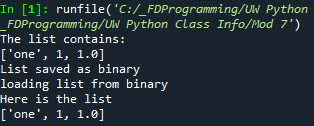
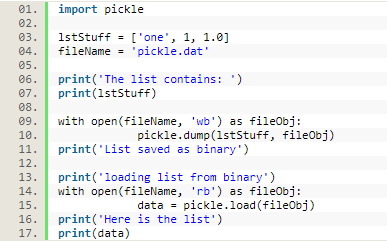


Figure 1 – Learning to Pickle

Binary files are identified by their ‘.dat’ extensions, and line four in Figure 1 above shows an example. Line nine is opening a binary file to write. Next, ‘lstStuff’ is saved to the binary file by using ‘pickle.dump()’. Line 14 is opening the binary file to read. Then, data is assigned the list that was previously saved by using ‘pickle.load()’. Notice how the list does not need to be processed or formatted, this is a benefit to using data files. Reference 1 has more information about pickling, I enjoyed how clear the examples were and that advertisements were kept to a minimum.

Structured error handling is the concept of building robustness into the program such that an error will not crash it. Anytime the program has to deal with an unknown, such as user input, it is worth evaluating if error handling is needed. The ‘try’ and ‘except’ statements are used to handle error cases. Figure two below shows an example of error handling when user input is being converted to an integer.

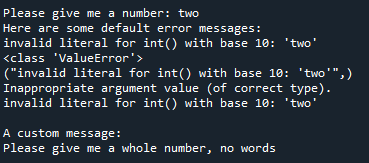
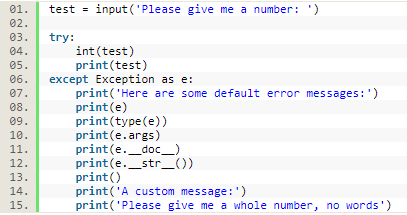


Figure 2 - Error Handling Options

When the program was executed the user chose to provide their number using a word rather than a number. Normally the program would terminate here; instead it executes the ‘except’ branch. Python has pre-built in errors, line 8-12 show some of the different options for displaying the error. As a programmer troubleshooting coding, I find lines 8, 9, and 11 the most useful because they highlight the problem. Line 9 shows the overall class that this error belongs to and results in “<class ‘ValueError’>”. Failing to convert to an integer is only one of the exceptions that live within the ‘ValueError’ class. If the program above converted to a float instead we would still see “<class ‘ValueError’>” as a result of line 9’s code. Line 11 shows the doc string used to describe this error class and results in “Inappropriate argument value (of correct type)”. Finally line 8 gives us the specific error generated within this class which results in “invalid literal for int() with base 10: ‘two’”. To summarize, lines 9 and 11 provide the big picture and line 8 gives the specifics. I think all of this information would be overwhelming on the user end so, I prefer to provide a custom message to the user to drive them to the desired input. It is also possible to create custom exceptions, Figure three below shows this.

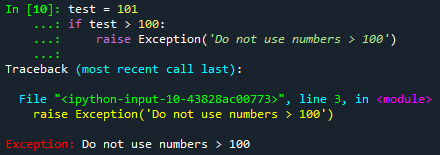


Figure 3 - Custom Error Generation

Another way to handle exceptions is by using the error type, Figure four shows an example of this.

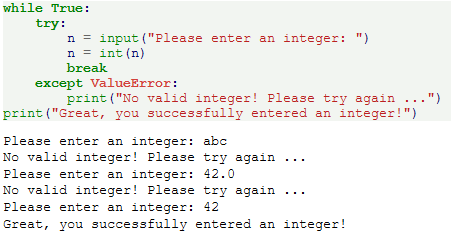


Figure 4 - Class Error Handling2

To learn more about error handling please review Reference 2. I felt the examples did a good job at covering multiple except clauses. Additionally, Reference 2 even talked about clean-up actions using the key word ‘finally’.

# Program: CD Inventory with Functions and Pickling

We were tasked with updating a program that would enable someone to save a digital record of their CDs, load the record from a file, and be able to delete specific saved data. The code for this can be seen in Appendix B.1. The updates required were: add structured error handling around areas of user interaction and modify data storage to use a binary file. Additionally, I incorporated comments and recommendations from last weeks’ assignment.

I noticed three key places for structured error handling. First, around the existence of the data file. I handled this possible error by checking if the file existed and then creating it if it did not. The second place for structured error handling was in the type conversion. This program expects the ID input from the user to be an integer. To accomplish this I included a ‘try’ except statement in ‘IO.new\_cd’ when converting the input to an integer. This ‘try’ and ‘except’ statement lives in a while loop so that the expected input is obtained prior to continuing. The final place I added error handling was around the import of the saved data. This was to protect against an empty saved data file. To accomplish the change in data storage, I changed the save file to binary. Next I imported the ‘pickle’ function and modified all file interactions to use ‘pickle.dump’ or ‘pickle.load’ to properly store the list of dictionaries as an object. Finally, I updated my functions based on last weeks’ feedback. Some of my function relied on the global variables to function correctly. It is preferred to have them operate independently and instead pass in any information, such as global variables, into them.

# Summary

I was successful in updating the CD Inventory program. I enjoyed learning how to ‘pickle’ objects, which makes processing information easier. The most difficult part this week was learning the preferred way to set up a function. Originally when I was learning about them my focus was narrowed to just the program I was in. I figured since they had access to the global scope of the program they didn’t need that information passed in. While this is true and will work, it is better to treat them as LEGO pieces you can move from one program to another. This means setting them up to expect information rather than just accessing the global version. When you do this it is easier to re-use work later.

# Appendix A.1

1. #------------------------------------------#
2. # Title: LAB07\_A.py
3. # Desc: simple demonstrator for classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # Brent Kieszling, 2020-Aug-22, Updated read and write file
7. #------------------------------------------#
9. # -- DATA -- #
10. strFileInput = 'mathIn.txt'
11. strFileOutput = 'mathOut.txt'
13. # -- PROCESSING -- #
14. **class** SimpleMath:
15. """A collection of simple math processing functions """
17. @staticmethod
18. **def** get\_sum(val1 = 0.0, val2 = 0.0):
19. """Function for adding two values

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

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

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

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

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

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

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

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

78. **class** IO:
79. """A collection of the Input / Output operations """
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. """
94. with open(fileName, 'r') as fileObj:
95. data = fileObj.read().strip().split(',')
96. numA = int(data[0])
97. numB = int(data[1])
98. **return** numA, numB
100. **def** write\_file(fileName, results):
101. """
102. function to write the math results to file fileName
104. Args:
105. fileName (string): file Name to write the results to.
106. results (list): The results
108. Returns:
109. None.
111. """
113. #convert each value in results to a string
114. a = 0
115. **for** value **in** results:
116. results[a] = str(results[a])
117. a += 1
119. with open(fileName, 'w') as fileObj:
120. fileObj.write(','.join(results) + '\n')
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)

# Appendix A.2

1. #------------------------------------------#
2. # Title: LAB07\_B.py
3. # Desc: simple demonstrator for classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # Brent Kieszling, 2020-Aug-22, Updated read and write file
7. #------------------------------------------#
9. **import** pickle
10. **import** os

13. # -- DATA -- #
14. strFileInput = 'numbers.dat'
15. strFileOutput = 'results.dat'
16. lstResults = []
17. # -- PROCESSING -- #
19. **if** os.path.exists(strFileInput) != True:
20. objFile = open(strFileInput, 'ab')
21. objFile.close()
23. **if** os.path.exists(strFileOutput) != True:
24. objFile = open(strFileOutput, 'ab')
25. objFile.close()
26. **class** Calc:
27. """A collection of simple math processing functions """
29. @staticmethod
30. **def** all\_calc(file\_Name):
31. """Function that performs the following operations: +, -, \*, /
33. This function processes the binary input from strFileInput. It expects
34. a list containing two numbers. It performs the 4 individual operations
35. (+, -, \*, /) and appends each result to lstResults and then saves the
36. list as binary in strFileOutput
38. Args:
39. file\_Name (string): Binary file
41. Returns:
42. None.
43. """
44. with open(file\_Name, 'rb') as fileObj:
45. lstInput = pickle.load(fileObj)
46. input1 = lstInput[0]
47. input2 = lstInput[1]
48. summ = input1 + input2
49. lstResults.append(summ)
51. diff = input1 - input2
52. lstResults.append(diff)
54. pro = input1 \* input2
55. lstResults.append(pro)
57. quo = input1 / input2
58. lstResults.append(quo)
60. with open(strFileOutput, 'wb') as fileObj:
61. pickle.dump(lstResults, fileObj)
63. @staticmethod
64. **def** get\_sum(val1 = 0.0, val2 = 0.0):
65. """Function for adding two values

68. Args:
69. val1: the first number to add
70. val2: the second number to add

73. Returns:
74. A float corresponding to the sum of val1 and val2
75. """
76. **return** float(val1 + val2)
78. @staticmethod
79. **def** get\_diffference(val1 = 0.0, val2 = 0.0):
80. """Function for subtracting two values

83. Args:
84. val1: the number to subtract from
85. val2: the number to subtract

88. Returns:
89. A float corresponding to the difference of val1 and val2
90. """
91. **return** float(val1 - val2)
93. @staticmethod
94. **def** get\_product(val1 = 0.0, val2 = 0.0):
95. """Function for multiplying two values

98. Args:
99. val1: the first number to multiply
100. val2: the second number to multiply

103. Returns:
104. A float corresponding to the product of val1 and val2
105. """
106. **return** float(val1 \* val2)
108. @staticmethod
109. **def** get\_quotient(val1 = 0.0, val2 = 0.0):
110. """Function for dividing two values

113. Args:
114. val1: the number to divide
115. val2: the number to divide by

118. Returns:
119. A float corresponding to the quotient of val1 and val2
120. """
121. **return** float(val1 / val2)

124. **class** IO:
125. """A collection of the Input / Output operations """
127. **def** read\_results(fileName):
128. """
129. function to read in resultss from file fileName
131. Args:
132. fileName (string): file name to read the numbers from
134. Returns:
135. None.
137. """
138. **try**:
139. with open(fileName, 'rb') as fileObj:
140. data = pickle.load(fileObj)
141. **print**('The results are: ')
142. **print**(data)
143. **except**:
144. **print**('No results found')

147. **def** get\_input(file\_Name):
148. """ A function to store two user inputs as a list in binary
150. Args:
151. file\_Name(string): file name to read the numbers from
153. Returns:
154. None.
156. """
157. **print**('Please provide two numbers to perform math on.')
158. numA = int(input('The first number: '))
159. numB = int(input('The second number: '))
160. lstInput = []
161. lstInput.append(numA)
162. lstInput.append(numB)
163. with open(file\_Name, 'wb') as fileObj:
164. pickle.dump(lstInput, fileObj)
165. **return**
167. **def** write\_file(fileName, results):
168. """
169. function to write the math results to file fileName
171. Args:
172. fileName (string): file Name to write the results to.
173. results (list): The results
175. Returns:
176. None.
178. """
180. with open(fileName, 'wb') as fileObj:
181. pickle.dump(results, fileObj)
183. # -- PRESENTATION (Input/Output) -- #
184. **print**('Basic Math script. Calculating the Sum, Difference, Product and Quotient of two numbers.')
185. IO.get\_input(strFileInput)
186. Calc.all\_calc(strFileInput)
187. IO.read\_results(strFileOutput)

# Appendix A.3

1. #------------------------------------------#
2. # Title: LAB07\_C.py
3. # Desc: simple demonstrator for classes
4. # Change Log: (Who, When, What)
5. # DBiesinger, 2030-Jan-01, Created File
6. # Brent Kieszling, 2020-Aug-22, Updated read and write file
7. #------------------------------------------#
9. **import** pickle
10. **import** os

13. # -- DATA -- #
14. strFileInput = 'numbers.dat'
15. strFileOutput = 'results.dat'
16. lstResults = []
17. # -- PROCESSING -- #
18. #This handles the possible error that would occur if file 'numbers.dat' did
19. #not exist. It does this by checking to see if the file exists and if
20. #it does not exist, it creats that file.
21. **if** os.path.exists(strFileInput) != True:
22. objFile = open(strFileInput, 'ab')
23. objFile.close()
25. #This handles the possible error that would occur if file 'results.dat' did
26. #not exist.
27. **if** os.path.exists(strFileOutput) != True:
28. objFile = open(strFileOutput, 'ab')
29. objFile.close()
30. **class** Calc:
31. """A collection of simple math processing functions """
33. @staticmethod
34. **def** all\_calc(fileName):
35. """Function that performs the following operations: +, -, \*, /
37. This function processes the binary input from fileName. It expects
38. a list containing two numbers. It performs the 4 individual operations
39. (+, -, \*, /) and appends each result to lstResults and then saves the
40. list as binary in strFileOutput
42. Args:
43. fileName (string): file name to read the numbers from.
45. Returns:
46. None.
47. """
48. with open(fileName, 'rb') as fileObj:
49. lstInput = pickle.load(fileObj)
50. input1 = lstInput[0]
51. input2 = lstInput[1]
52. summ = input1 + input2
53. lstResults.append(summ)
55. diff = input1 - input2
56. lstResults.append(diff)
58. pro = input1 \* input2
59. lstResults.append(pro)
61. quo = input1 / input2
62. lstResults.append(quo)
64. with open(strFileOutput, 'wb') as fileObj:
65. pickle.dump(lstResults, fileObj)
66. **return**

69. **class** IO:
70. """A collection of the Input / Output operations """
72. **def** read\_results(fileName):
73. """
74. function to read in resultss from file fileName
76. Args:
77. fileName (string): file name to read the numbers from
79. Returns:
80. None.
82. """
83. with open(fileName, 'rb') as fileObj:
84. data = pickle.load(fileObj)
85. **print**('The results are: ')
86. **print**(data)


90. **def** get\_input(file\_name):
91. """ A function to store two user inputs as a list in binary
93. Args:
94. fileName (string): file name to save the numbers to.
96. Returns:
97. None.
99. """
100. **print**('Please provide two numbers to perform math on.')
101. numA = input('The first number: ')
102. #This handles the case if the user enters anything other than an integer
103. **while** True:
104. **try**:
105. numA = int(numA)
106. **break**
107. **except**:
108. numA = input('Please enter a numerical integer. ')

111. #This checks to see if numB is 0 and prompts the user to enter a different
112. #number if it is. Also verifies the input is an integer
113. **while** True:
114. numB = input('The second number: ')
115. **try**:
116. numB = int(numB)
117. numA /numB
118. **break**
119. **except** ValueError:
120. **print**('Please enter a numerical integer.')
121. **except** ZeroDivisionError:
122. **print**('Please enter a non 0 integer.')

125. lstInput = []
126. lstInput.append(numA)
127. lstInput.append(numB)
128. with open(file\_name, 'wb') as fileObj:
129. pickle.dump(lstInput, fileObj)
130. **return**
132. **def** write\_file(fileName, results):
133. """
134. function to write the math results to file fileName
136. Args:
137. fileName (string): file Name to write the results to.
138. results (list): The results
140. Returns:
141. None.
143. """
145. with open(fileName, 'wb') as fileObj:
146. pickle.dump(results, fileObj)
148. # -- PRESENTATION (Input/Output) -- #
149. **print**('Basic Math script. Calculating the Sum, Difference, Product and Quotient of two numbers.')
150. IO.get\_input(strFileInput)
151. Calc.all\_calc(strFileInput)
152. IO.read\_results(strFileOutput)

# Appendix B.1

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. #Brent Kieszling, 2020-Aug-17, Added function new\_cd and add\_cd
7. #Brent Kieszling, 2020-Aug-18, Updated function write\_file
8. #Brent Kieszling, 2020-Aug-19, Fixed
9. #Brent Kieszling, 2020-Aug-24, Updated Data Processor and File Processor functions,
10. #       converted from .txt to .dat file type, Added error handling,
11. #------------------------------------------#
13. **import** os
14. **import** pickle
16. # -- DATA -- #
17. strChoice = '' # User input
18. lstTbl = []  # list of lists to hold data
19. dicRow = {'ID': '', 'Title': '', 'Artist': ''}  # list of data row
20. strFileName = 'CDInventory.dat'  # data storage file
21. objFile = None  # file object
23. # -- PROCESSING -- #
24. **if** os.path.exists(strFileName) != True:
25. objFile = open(strFileName, 'ab')
26. objFile.close()
27. **class** DataProcessor:
28. """Processes the data"""
29. @staticmethod
30. **def** add\_cd(tplNewCD, lstCurrentCDs):
31. """Adds a new CD
33. Takes the user input tplNewCD and assigns each value to a dictionary with keys:
34. ID, Title, and Artist. Then it adds it to the active table.
36. Args:
37. tplNewCD (tuple): Contains 3 items: ID, Title, and Artist.
38. lstCurrentCDs (list): A list of dictionaries all with the keys:
39. 'ID', 'Tittle', and 'Artist'
41. Returns:
42. lstCurrentCDs(list): Returns the updated list after adding the new entry
43. """
44. dicNewCD = {'ID': '', 'Title': '', 'Artist': ''}
45. dicNewCD = {'ID': tplNewCD[0], 'Title': tplNewCD[1], 'Artist': tplNewCD[2]}
46. lstCurrentCDs.append(dicNewCD)
47. **return** lstCurrentCDs
49. @staticmethod
50. **def** delete\_cd(remove\_ID, lstCurrentCDs):
51. """Deletes a CD
53. Takes the user input remove\_ID and searches the active table for the
54. appropriate ID and removes it. Additionaly, tracks success
55. via blnCDRemoved.
57. Args:
58. remove\_ID (interger): Holds the ID the user requested be removed.
60. Returns:
61. blnCDRemoved(boolean): True if row removed otherwise False.
62. lstCurrentCDs(list): Returns the updated list after adding the new entry
63. """
64. intRowNr = -1
65. blnCDRemoved = False
66. **for** row **in** lstCurrentCDs:
67. intRowNr += 1
68. **if** row['ID'] == intIDDel:
69. **del** lstCurrentCDs[intRowNr]
70. blnCDRemoved = True
71. **break**
72. **return** blnCDRemoved, lstCurrentCDs


76. **class** FileProcessor:
77. """Processing the data to and from text file"""
79. @staticmethod
80. **def** read\_file(file\_name, table):
81. """Function to import a list of dictionaries (lstSavedCollection) from a binary file.
83. Args:
84. file\_name (string): name of file used to read the data from
85. table (list): 2D table (list of dicts)
87. Returns:
88. table (list): 2D table (list of dicts)
89. """
90. table.clear()
91. #The try statement handles an instance where there is no saved data
92. **try**:
93. with open(file\_name, 'rb') as fileObj:
94. table = pickle.load(fileObj)
95. **except**:
96. **pass**
97. **return** table
99. @staticmethod
100. **def** write\_file(file\_name, table):
101. """Function to save a list of dictionaries (table) to a binary file
103. Args:
104. file\_name (string): name of file used to read the data from
105. table (list): 2D table (list of dicts)
107. Returns:
108. None.
109. """
110. with open(file\_name, 'wb') as fileObj:
111. pickle.dump(table, fileObj)

114. # -- PRESENTATION (Input/Output) -- #
116. **class** IO:
117. """Handling Input / Output"""
119. @staticmethod
120. **def** print\_menu():
121. """Displays a menu of choices to the user
123. Args:
124. None.
126. Returns:
127. None.
128. """
130. **print**('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display Current Inventory')
131. **print**('[d] delete CD from Inventory\n[s] Save Inventory to file\n[x] exit\n')
133. @staticmethod
134. **def** menu\_choice():
135. """Gets user input for menu selection
137. Args:
138. None.
140. Returns:
141. choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x
143. """
144. choice = ' '
145. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
146. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
147. **print**()  # Add extra space for layout
148. **return** choice
150. @staticmethod
151. **def** show\_inventory(table):
152. """Displays current inventory table

155. Args:
156. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime.
158. Returns:
159. None.
161. """
162. **print**('======= The Current Inventory: =======')
163. **print**('ID\tCD Title (by: Artist)\n')
164. **for** row **in** table:
165. **print**('{}\t{} (by:{})'.format(\*row.values()))
166. **print**('======================================')
168. @staticmethod
169. **def** new\_cd():
170. """Allows the user to add a CD to the active inventory table
172. Args:
173. None.
175. Returns:
176. intID (interger): Serialized ID
177. strTitle (string): Tittle of CD
178. stArtist (string): Name of artist
180. """
181. # 3.3.1 Ask user for new ID, CD Title and Artist
182. strID = input('Enter ID: ').strip()
183. **while** True:
184. #This try handles the case where a non interger is entered
185. **try**:
186. intID = int(strID)
187. **break**
188. **except**:
189. strID = input('Please enter an interger for the ID.')
190. strTitle = input('What is the CD\'s title? ').strip()
191. stArtist = input('What is the Artist\'s name? ').strip()
192. **return** intID, strTitle, stArtist

195. # 1. When program starts, read in the currently saved Inventory.
196. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
198. # 2. start main loop
199. **while** True:
200. # 2.1 Display Menu to user and get choice
201. IO.print\_menu()
203. # 3. Process menu selection
204. strChoice = IO.menu\_choice()
205. # 3.1 process exit first
206. **if** strChoice == 'x':
207. **break**
208. # 3.2 process load inventory
209. **if** strChoice == 'l':
210. **print**('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
211. strYesNo = input('type \'yes\' to continue and reload from file. otherwise reload will be canceled. ')
212. **if** strYesNo.lower() == 'yes':
213. **print**('reloading...')
214. lstTbl = FileProcessor.read\_file(strFileName, lstTbl)
215. IO.show\_inventory(lstTbl)
216. **else**:
217. input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
218. IO.show\_inventory(lstTbl)
219. **continue**  # start loop back at top.
220. # 3.3 process add a CD
221. **elif** strChoice == 'a':
222. #This casts the return from IO.new\_cd() into the function DataProcessor.add\_cd
223. lstTbl = DataProcessor.add\_cd(IO.new\_cd(), lstTbl)
224. IO.show\_inventory(lstTbl)
225. **continue**  # start loop back at top.
226. # 3.4 process display current inventory
227. **elif** strChoice == 'i':
228. IO.show\_inventory(lstTbl)
229. **continue**  # start loop back at top.
230. # 3.5 process delete a CD
231. **elif** strChoice == 'd':
232. # 3.5.1 get Userinput for which CD to delete
233. # 3.5.1.1 display Inventory to user
234. IO.show\_inventory(lstTbl)
235. # 3.5.1.2 ask user which ID to remove
236. **while** True:
237. strIDDel = input('Which ID would you like to delete? ').strip()
238. **try**:
239. intIDDel = int(strIDDel)
240. **break**
241. **except**:
242. **print**('Please enter an integer.')
243. # 3.5.2 search thru table and delete CD
244. blnCDRemoved, lstTbl = DataProcessor.delete\_cd(intIDDel, lstTbl)
245. **if** blnCDRemoved:
246. **print**('The CD was removed')
247. **else**:
248. **print**('Could not find this CD!')
249. IO.show\_inventory(lstTbl)
250. **continue**  # start loop back at top.
251. # 3.6 process save inventory to file
252. **elif** strChoice == 's':
253. # 3.6.1 Display current inventory and ask user for confirmation to save
254. IO.show\_inventory(lstTbl)
255. strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
256. # 3.6.2 Process choice
257. **if** strYesNo == 'y':
258. # 3.6.2.1 save data
259. FileProcessor.write\_file(strFileName, lstTbl)
260. **else**:
261. input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
262. **continue**  # start loop back at top.
263. # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
264. **else**:
265. **print**('General Error')

# Appendix B.2

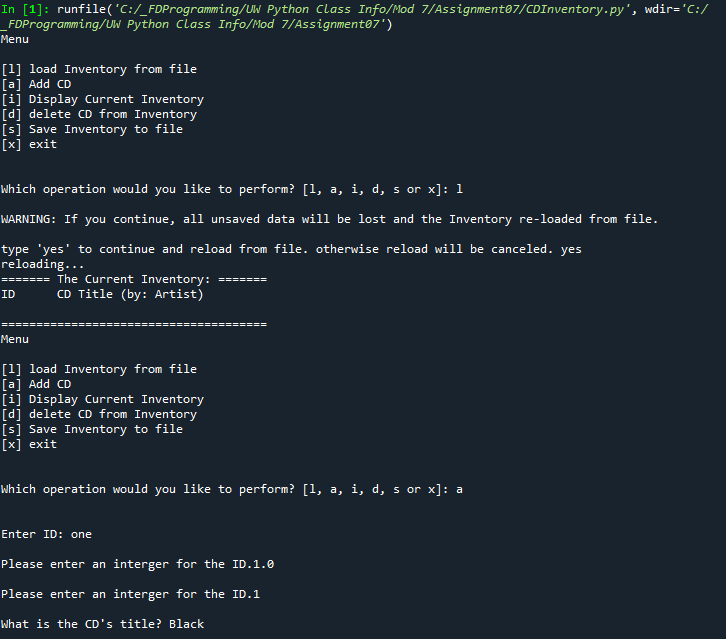


Figure 5 - Program run in Spyder part 1/3

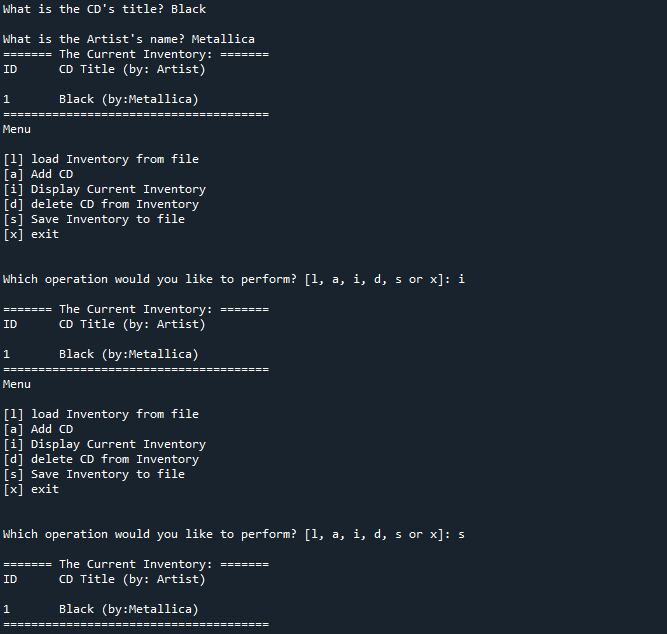


Figure 6 - Program run in Spyder part 2/3

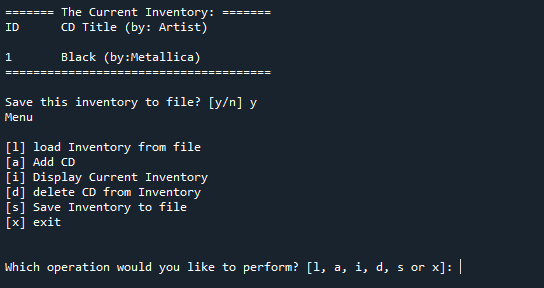


Figure 7 - Program run in Spyder part 3/3

# Appendix B.3

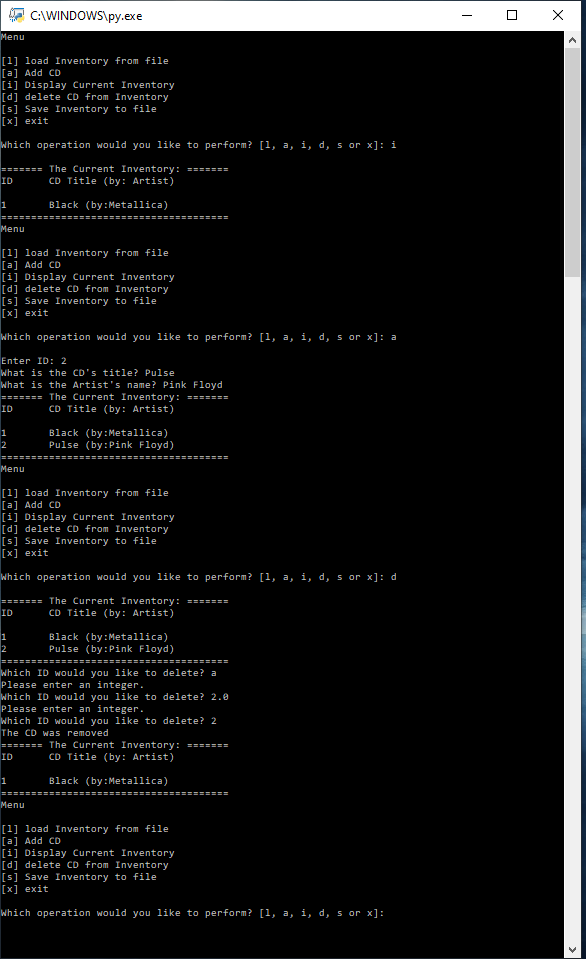


Figure 8 – Program run in Terminal

# GitHub Link

1. https://github.com/Brent-K/Assignment\_07

# References

1. <https://www.pitt.edu/~naraehan/python3/pickling.html> Accessed: 8/26/2020
2. https://www.python-course.eu/python3\_exception\_handling.php Accessed: 8/26/2020