Beemnet Workeneh

03/02/20

FDN 100

Assignment 07

TITLE

# Introduction

In this assignment we will learn the benefits of using structured error handling, the differences between a text file and a binary file, the usage of the Exception class, , the “derivation” of a new class from Exception class and when to do it, and Markdown language. We will also research exception handling and pickling and document our findings. We will use some of what we’ve learned to then enhance the CD Inventory script that we have been working on for the past assignments.

# Exploration Questions

*What are the benefits of using structured error handling?*

If we have structured exception handling functionality the program won’t end abruptly and at the very least it can “exit gracefully instead of crashing awkwardly”[[1]](#footnote-1). We can “trap” for specific type of exception such as IndexError or ValueError and point out exactly how the Exception was raised. It is a good idea to trap for exceptions when opening a file or reading in case the file doesn’t exist. It’s helpful when the script has a user input portion to catch and inform the type of exception that was caught and handle each exception type individually. Using the *finally* block along with the *try* and *except* blocks allows for closing a file that was opened inside the try block and could not be closed because an exception was thrown[[2]](#footnote-2). Structured error handling allows you to save data memory that would otherwise be lost if the program crashes.

*What are the differences between a text file and a binary file?*

A text file only stores characters. A text file is human readable. It requires the data to be formatted in order to save the text file. A binary file can store entire complex piece of data. For example using the pickle module in python we can write entire objects such as numbers, strings, tuples, lists, and dictionaries—these data types do not have to be formatted to be saved as characters. When we read a text file we retrieve characters line by line, when we read binary files we can load entire objects at once. Binary is helpful for saving states of your application or settings the user made.

*How is the Exception class used?*

The Exception class is part of the BaseException. The Exception class has a list of specialized exceptions such as IndexError or ValueError. The Exception class is used as a base class when we define our own custom Exception classes. Derived classes inherit from the base class. The data and functions can be overwritten end extended in the derived class for our specific needs.

*How do you "derive" a new class from the Exception class?*

We “derive” a new class from the Exception class the following way:

class DerivedExceptionName (Exception):

“ “ “ docstring “ “ “

def \_str\_(self):

return “Custome exception message”

We define a class with our custom derived exception class name and including “Exception” in the parenthesis. We get our own custom string dunder function \_str\_.

*When might you create a class derived from the Exception class?*

We might want to create a derived Exception class when we want specialized Exceptions than what already exist in the Exception Class. For example we can create an exception for when a person enters a value that is not within a specified range (i.e “the value entered is negative please enter a positive number” or “the value entered is too high, please enter a value below such and such”).

*What is the Markdown language?*

Markdown is (1) a plain text formatting syntax; (2) a software tool, written in Perl, that converts plain test formatting to HTML[[3]](#footnote-3). The goal for Markdown’s formatting syntax is to make it as readable as possible. It is influenced by the format of plain text email. The readme file’s styling is simplified by using markdown code.

*Exception Handling in Python: web example*

The website w3schools.com has simple and easy to understand examples of exception handling in python2. Unlike chapter 7 of the text book it also introduces the *finally* block which is used to execute the block regardless of having an exception trapped or not. The website also gives example of *raise* keyword which the programmer can use to raise a specific type of exception based on a condition.

*Pickling in Python: web example*

The website datacamp.com has a good example of how to use pickling in python[[4]](#footnote-4). Much like the text book it explains the concept of pickling, which types of data types can be pickled, and how to pickle and unpickle by using simple examples comprised of very few lines. Unlike the book it delves into when to not use pickling, as well as comparing pickling vs JSON. It also talks about multiprocessing and compressing of pickle files which seem out of the scope of this project but may become important in the future.

# CDInventory Program

In this part of the assignment we modify the CDInventory program from last week to include Exception handling.

I identified two locations where exception handling needed to be added: add\_cd() function and read\_file() function.

**The add\_cd() function:**

I modified the add\_cd() function from the previous assignment to catch if a value that cannot be type cast to an integer is entered by the user for the ID value of the CD inventory (also updated the docstring accordingly). After catching the exception, the program prompts the user to enter an integer. If the user does enter an integer the program uses the ID entered to update the inventory. If the person enters a non-integer when prompted by the exception handling block, the program crashes.

1. **def** add\_cd(table):
2. """ function to add a CD to the inventory
3. Tells user to enter an integer if a value that cannot be type cast to
4. integer is entered
5. Args:
6. table: the list of dictionaries containing the CD entries
8. Returns:
9. the modified list of dictionaries with new etries of CDs
10. """
12. **try**:
13. strID = input('Enter ID: ').strip()
14. intID = int(strID)
16. **except** ValueError as e:
17. **print**('<<< custom error message \nThat is not an integer!')
18. **print**('Built in error info:')
19. **print**(type(e), e, e.\_\_doc\_\_, sep= '\n')
20. **print**('\n Enter an integer for the ID: ')
21. strID = input('Enter ID: ').strip()
22. intID = int(strID)
23. #it is possible to add custom exception handling for repeated CD ID
24. strTitle = input('What is the CD\'s title? ').strip()
25. stArtist = input('What is the Artist\'s name? ').strip()
26. # Add item to the table
27. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
28. table.append(dicRow)
29. **return** table

Listing 1 add\_cd() function with exception handling

|  |  |
| --- | --- |
| Executed in spyder | Executed in anaconda prompt |
|  |  |

Table Catching an Exception inside the add\_cd() function

**The read\_file() function**

I modified the read\_file() function to trap if a FileNotFoundError is thrown (also updated the docstring accordingly) . If a FileNotFoundError is caught, the program creates the specified file by opening the function using write mode. I believe it is also possible to substitute two lines for the entire FileNotFoundError Exception handling(opening the file using write mode, then closing the file object before reopening it with read mode). However doing it this way would not be informative.

1. **def** read\_file(file\_name, table):
2. """Function to manage data ingestion from file to a list of dictionaries
4. Reads the data from file identified by file\_name into a 2D table
5. (list of dicts) table one line in the file represents one dictionary row in table.
6. If the file does not exist it creates it by using Exception handling
8. Args:
9. file\_name (string): name of file used to read the data from
10. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
12. Returns:
13. None.
14. """
15. table.clear()  # this clears existing data and allows to load data from file
16. **try**:
17. objFile = open(file\_name, 'r')
18. **except** FileNotFoundError as e:
19. **print**('<<< custom error message \n the file is not found')
20. **print**('Built in error info:')
21. **print**(type(e), e, e.\_\_doc\_\_, sep= '\n')
22. **print**('Creating ', file\_name)
23. objFile = open(file\_name, 'w')# create the file
24. objFile.close()
25. objFile = open(file\_name, 'r') # re-opening incase there was a FileNotFoundError
26. **for** line **in** objFile:
27. data = line.strip().split(',')
28. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
29. table.append(dicRow)
30. objFile.close()

Listing 2 the read\_file() function with exception handling

|  |  |
| --- | --- |
| Executed in spyder | Executed in anaconda prompt |
|  |  |

Table Catching a FileNotFoundError inside the read\_file() function

I did not think that the menu\_choice() function demanded exception handling because the way it is currently written if a value entered is not in the list of appropriate characters, the while loop will continue to churn and ask the user to enter a value from the displayed list.

I did not think that show\_inventory() function needed exception handling in the case that an empty table is fed to it. If an empty table is fed to it, it does not throw an exception, it will just display an empty inventory.

# Summary

# Appendix

## GitHub Repository

The CDInventory code can be found under the following link

https://github.com/beemnet20/Assignment\_07

## CDInventory.py

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. # BWorkeneh, 2020-Feb-26, completed the TODOs in the starter code
7. # BWorkeneh, 2020-Mar-4, added Exception Handling
8. #------------------------------------------#
10. # -- DATA -- #
11. strChoice = '' # User input
12. lstTbl = []  # list of lists to hold data
13. dicRow = {}  # list of data row
14. strFileName = 'CDInventory.txt'  # data storage file
15. objFile = None  # file object

18. # -- PROCESSING -- #
19. **class** DataProcessor:
20. """ Processing the information in memory """
21. @staticmethod
22. **def** add\_cd(table):
23. """ function to add a CD to the inventory
24. Tells user to enter an integer if a value that cannot be type cast to
25. integer is entered
26. Args:
27. table: the list of dictionaries containing the CD entries
29. Returns:
30. the modified list of dictionaries with new etries of CDs
31. """
33. **try**:
34. strID = input('Enter ID: ').strip()
35. intID = int(strID)
37. **except** ValueError as e:
38. **print**('<<< custom error message \nThat is not an integer!')
39. **print**('Built in error info:')
40. **print**(type(e), e, e.\_\_doc\_\_, sep= '\n')
41. **print**('\n Enter an integer for the ID: ')
42. strID = input('Enter ID: ').strip()
43. intID = int(strID)
44. #it is possible to add custom exception handling for repeated CD ID
45. strTitle = input('What is the CD\'s title? ').strip()
46. stArtist = input('What is the Artist\'s name? ').strip()
47. # Add item to the table
48. dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}
49. table.append(dicRow)
50. **return** table
51. @staticmethod
52. **def** delete\_cd(table, ID):
53. """ function to delete an entry from the inventory
55. Args:
56. table: the list of dictionaries containing the CD entries
57. ID: the integer ID of the CD to be deleted
59. Returns:
60. the modified list of dictionaries with the entry containing the ID removed
61. """
63. # search thru table and delete CD
64. intRowNr = -1
65. blnCDRemoved = False
66. **for** row **in** table:
67. intRowNr += 1
68. **if** row['ID'] == ID:
69. **del** table[intRowNr]
70. blnCDRemoved = True
71. **break**
72. **if** blnCDRemoved:
73. **print**('The CD was removed')
74. **else**:
75. **print**('Could not find this CD!')
76. **return** table
78. **class** FileProcessor:
79. """Processing the data to and from text file"""
81. @staticmethod
82. **def** read\_file(file\_name, table):
83. """Function to manage data ingestion from file to a list of dictionaries
85. Reads the data from file identified by file\_name into a 2D table
86. (list of dicts) table one line in the file represents one dictionary row in table.
87. If the file does not exist it creates it by using Exception handling
89. Args:
90. file\_name (string): name of file used to read the data from
91. table (list of dict): 2D data structure (list of dicts) that holds the data during runtime
93. Returns:
94. None.
95. """
96. table.clear()  # this clears existing data and allows to load data from file
97. **try**:
98. objFile = open(file\_name, 'r')
99. **except** FileNotFoundError as e:
100. **print**('<<< custom error message \n the file is not found')
101. **print**('Built in error info:')
102. **print**(type(e), e, e.\_\_doc\_\_, sep= '\n')
103. **print**('Creating ', file\_name)
104. objFile = open(file\_name, 'w')# create the file
105. objFile.close()
106. objFile = open(file\_name, 'r') # re-opening incase there was a FileNotFoundError
107. **for** line **in** objFile:
108. data = line.strip().split(',')
109. dicRow = {'ID': int(data[0]), 'Title': data[1], 'Artist': data[2]}
110. table.append(dicRow)
111. objFile.close()

114. @staticmethod
115. **def** write\_file(file\_name, table):
116. """Function to write data that is in a list of dictionaries to a text file
118. Reads the data from the list of dictionaries and writes into a file identified by file\_name ad
119. rows of comma delimited items.
121. Args:
122. file\_name(string): name of file used to write the data to
123. table( list of dict): 2D data structure (list of dicts) that holds the data during runtime
125. Returns:
126. None.
127. """
128. # write mode used to open the file, will create the file if it is not found, no exception hadndling necessary
129. objFile = open(file\_name, 'w')
130. **for** row **in** table:
131. lstValues = list(row.values())
132. lstValues[0] = str(lstValues[0])
133. objFile.write(','.join(lstValues) + '\n')
134. objFile.close()


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 string of the users input out of the choices l, a, i, d, s or x
167. """
168. choice = ' '
169. # the condition of the while loop below is sort of a custom exception handling
170. **while** choice **not** **in** ['l', 'a', 'i', 'd', 's', 'x']:
171. choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
172. **print**()  # Add extra space for layout
173. **return** choice
175. @staticmethod
176. **def** show\_inventory(table):
177. """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**('======================================')

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

1. Dawson, Michael. Python® Programming for the Absolute Beginner, Third Edition. Course Technology PTR, 2009. [↑](#footnote-ref-1)
2. <https://www.w3schools.com/python/python_try_except.asp> (retrieved 3/2/20) [↑](#footnote-ref-2)
3. <https://daringfireball.net/projects/markdown/> (retrieved 3/3/20) [↑](#footnote-ref-3)
4. <https://www.datacamp.com/community/tutorials/pickle-python-tutorial> (posted 4/5/18, retrieved 3/2/20) [↑](#footnote-ref-4)