Ann Allen

Introduction to Programming – Python

Assignment 07

August 22, 2020

Pickling and Structured Error Handling

Introduction

This assignment returns to the CDInventory.py script written for Assignment 6. The script is updated to use a binary data format for saving inventory data. This data is written and accessed using the Python pickle module. I also learned more robust error handling and messaging and added try-except error handling throughout the program. The results are posted to GitHub.

GitHub repository

This document and the Assignment 7 CDInventory.py script are posted at this GitHub repository:
https://github.com/annspiral/Assignment_07

Pickling

Pickling and unpickling in Python is a way to save and access content in binary format. Using the pickle module in Python, content as simple as a string and as complex as a dictionary or a class can be stored in a file. Pickling and unpickling is a serializing and deserializing process. Serializing and deserializing processes may be referred to as marshaling/unmarshalling or flattening in other languages.[1] Although a similar action is taking place when pickling in Python or marshaling in another language, Python.org cautions that when using the Python pickle binary format it is best to refer to it specifically as pickling and not marshaling or flattening.[2]

Pickled data file vs. Text file

When we saved data in a text file, we were only able to read and write content as strings. With the pickle module we can save objects without unpacking them into strings. There are both advantages and disadvantages of these options.

- Text Files are cross platform

[1] RealPython, https://realpython.com/python-pickle-module/, August 22, 2020
[2] Python.org, https://docs.python.org/3/library/pickle.html, August 22, 2020

    - Text files can be used on a Window, Mac or Unix system and there are tools easily available to view and create them[3]
- Text files can be easily created for program input
    - The program in Chapter 7 of our course book also demonstrates the value of a text file used to provide program input. New episodes of quiz questions can be added to the program easily with new text files.
- Pickled files can save memory
    - Modifying content to save as strings and convert strings back into their object format can use extra memory[4] and can also be slow[5].
- Pickled files can store more complex content
    - Pickle files can store lists, dictionaries, classes and store their structure as well as their contents as their defined value type.[6]

Pickling without a data file

When using pickling in Python, there are two main options – `dump()` and `load()` or and `dumps()` and `loads()`. Using `dump()` and `load()` the objects are saved and read from a file. Using `dumps()` and `loads()`, objects can be serialized into a string that can be passed as a variable. This variable can be converted back from the byte stream string using `loads()`. This byte stream string can then be compressed if size is a main concern.[7]

Pickling security

Our course materials, the course book and almost every site that I explored were all clear to mention that pickling is *not* encrypting. While the content is not easily human readable the characters can also be easily interpreted knowing the pickling format.

Some of the websites that I referenced also flagged the risk of unpickling a file from an unknown source. RealPython described a scenario where the `__setstate__()` method of an object can be used to "execute arbitrary code during the unpickling process!"[8]

Pickle used in CDInventory.py

In the CDInventory.py script, I use the imported pickle module to `load()` and `dump()` the CDInventory 2D list to file.

Pickle.load()

To load in the CDInventory list in Assignment 6, I had to read in the strings of the text file, trust the comma separators and rebuild the 2D list. For large sets of this might be memory intensive.

[3] Dawson, Michael, Python Programming 3rd Edition, Course Technology, 2010
[4] Biesinger, Dirk, Introduction to Programming – Python, FDN_Py_Module_07.pdf, August 22, 2020
[5] Pythonprogramming, https://pythonprogramming.net/python-pickle-module-save-objects-serialization/, August 24, 2020
[6] Python.org, https://docs.python.org/3.7/library/pickle.html, August 24, 2020
[7] Geeksforgeeks, https://www.geeksforgeeks.org/pickle-python-object-serialization/?ref=lbp, August 24, 2020
[8] RealPython, https://realpython.com/python-pickle-module/, August 22, 2020

The conversion also creates complexity to ensure that each value goes through the conversion to string and back again. Using `pickle.load()` for this assignment, I was able to load the 2D list of dictionaries directly and the copy it into the script variable with no string converting or restructuring (Listing 1). `Pickle.load()` on line 252 of Listing 1, returns a list object, which I am then able to copy into `table` as an in-memory inventory table.

```
178.    @staticmethod
179.        def read_file(file_name, table):
180.            """Function to manage data ingestion from file to a list of dictionaries
181.     …

194.            # clear table to load with file contents
195.            table.clear()
196.            try:
197.                # open file to read binary
198.                objFile = open(file_name, 'rb')
199.            except FileNotFoundError as e:
200.                print('! FileNotFoundError reading file: ', e ,'\n')
201.                return False
202.            except PermissionError as e:
203.                print('! PermissionError reading file: ', e ,'\n')
204.                return False
205.            except Exception as e:
206.                print('! Error reading file: ', e ,'\n')
207.                return False
208.            else:
209.                try:
210.                    # copy contents of file into table
211.                    table[:] = pickle.load(objFile)[:]
212.                except AttributeError as e:
213.                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
214.                          'Recommended to check data file for corruption.')
215.                    return False
216.                except pickle.UnpicklingError as e:
217.                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
218.                          'Recommended to check data file for corruption.')
219.                    return False
220.                except EOFError as e:
221.                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
222.                          'Recommended to check data file for corruption.')
223.                    return False
224.                except Exception as e:
225.                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
226.                          'Recommended to check data file for corruption.')
227.                    return False
228.                objFile.close()
229.                return True
```

Listing 1: CDInventory.py listing showing use of pickle.load() to load inventory from file as object

Note that when using the `pickle.load()` method, the file accessed needs to be opened for reading specifically in binary format. On line 239 in Listing 1, I opened the CDInventory.dat file using the “rb” mode. This mode supports reading a binary file.

Pickle.dump()

To save the CDInventory list in Assignment 6, all the contents of the list had to be unpacked and converted in comma separated strings. For a large data set, this would consume a large amount of memory. By using `pickle.dump()` in this assignment, I was able to pass the inventory list object directly (Listing 2, Line 264). The contents of the inventory list along with all the structure of the object is then stored in the CDInventory.dat file.

```
233.        @staticmethod
234.        def write_file(file_name, table):
235.            """Function to write updated inventory list of dictionaries to file
236.     …
237.            """
238.

252.            try:
253.                # open file to write binary
254.                objFile = open(file_name, 'wb')
255.            except PermissionError as e:
256.                print('! PermissionError reading file: ', e ,'\n')
257.                return False
258.            except Exception as e:
259.                print('! Error reading file: ', e ,'\n')
260.                return False
261.            else:
262.                try:
263.                    # add table data to file
264.                    pickle.dump(table, objFile)
265.                except AttributeError as e:
266.                    print('! Error pickling data file: ', e, '\n')
267.                    return False
268.                except pickle.PicklingError as e:
269.                    print('! Error pickling data file: ', e ,'\n')
270.                    return False
271.                except Exception as e:
272.                    print('! Error pickling data file: ', e ,'\n')
273.                    return False
274.                objFile.close()
275.                return True
```

Listing 2: CDInventory.py code showing pickle.dump() used to save data as a list object to file

Note that when using the `pickle.dump()` method, the file accessed needs to be opened for reading specifically in binary format. On line 254 in Listing 2, I opened the CDInventory.dat file using the "wb" mode. This mode supports reading a binary file.

Researching Pickling and Unpickling

The first site I checked for more information on pickling was RealPython at this site https://realpython.com/python-pickle-module/. This site seemed directed at a more experienced Python developer that may need already juggle different types of serializing and different Python versions. Looking for more of an intro or tutorial, I found TutorialsPoint, https://www.tutorialspoint.com/python-pickling. This site and the page by Pythonprogramming.com at https://pythonprogramming.net/python-pickle-module-save-objects-serialization/, provided some simple introductory examples. I particularly like the real world example on the Pythonprogramming.com page of using pickling to save machine learning states. This site helped to better understand pickling as a programming resource.

Structured Error Handling

Structured error handling allows a program to catch most errors and continue, rather than abruptly stopping. In cases of user input, the program can ask the user again if previous input created an error. If loading a file fails, the program can notify the user and the user might be able to fix the problem. If a program is handling multiple scenarios, it would be possible to still present the solutions to most of them even if only one fails.

Using Try-Except-Else-Finally

Using the `try-except` construct in Python, any line or lines of code can be tried in a `try` block to see if they succeed. If the code cannot be executed, Python will jump to the `except` block rather than ending the program and passing the system error message. If the code runs without error, the program continues and skips any `except` blocks.

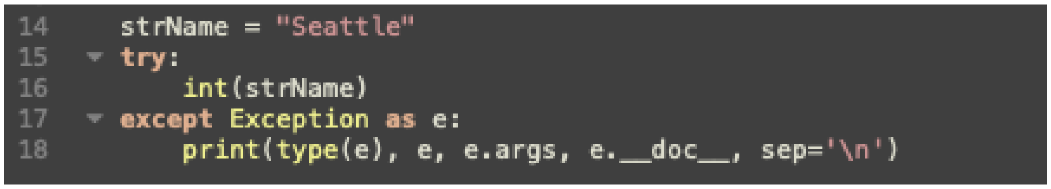
As Dirk Biesinger said in the course video module, “In the try block we are encapsulating the code block we want to secure against errors.”[9]

The `try-except` construct also includes `else` and `finally` options. Using `else`, if the code in the `try` block succeeds, the `else` is also executed. `Else` is not executed if the `except` block is used.[10]

If a `try-except` construct includes a `finally` option, this code is executed if the `try` is successful or even if it is not. Even after processing an `except` block, the code in `finally` is also executed.[11]

Using Exception class

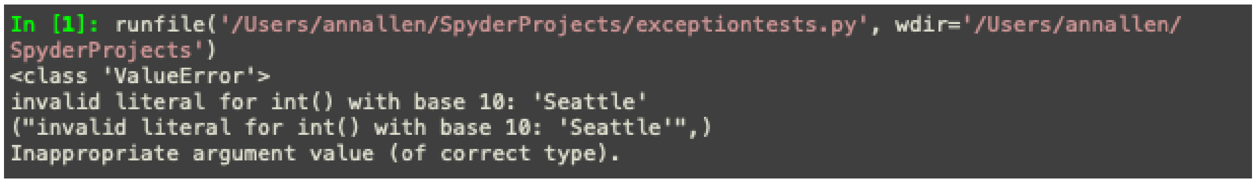
The Exception class is the general error or exception handling class. Using this class, you can retrieve – or “catch” - the information about the specific error that may have occurred. As seen in Figures 1 and 2 the `except` case can catch the Exception class and assign the exception to an object. This object then has methods to call to retrieve the specific exception details.



```
strName = "Seattle"
try:
    int(strName)
except Exception as e:
    print(type(e), e, e.args, e.__doc__, sep='\n')
```

Figure 1: Example code showing how to use the Exception class

[9] Biesinger, Dirk, Introduction to Programming – Python, Module 7-Video 3, https://www.youtube.com/watch?v=5_HaN4ggNAg&feature=youtu.be, August 22, 2020
[10] Dawson, Michael, Python Programming 3rd Edition, Course Technology, 2010
[11] Schafer, Corey, https://www.youtube.com/watch?v=NIWwJbo-9_8, August 23, 2020



```
In [1]: runfile('/Users/annallen/SpyderProjects/exceptiontests.py', wdir='/Users/annallen/
SpyderProjects')
<class 'ValueError'>
invalid literal for int() with base 10: 'Seattle'
("invalid literal for int() with base 10: 'Seattle'",)
Inappropriate argument value (of correct type).
```

Figure 2: Output from code in Figure 1, showing exception information

Using specific error cases

While the Exception class can catch any error and then map it to its specific error object, a specific error can be handled directly. In our course textbook, it states that it is "good programming practice to specify exception types so that you can handle each individual case. In fact, it's dangerous to catch all exceptions…. Generally, you should avoid that type of catchall."[12] This sentiment is reinforced in the python.org documentation.[13]

During our Week 8 course meeting, our course instructor clarified that the Exception class should not be used as the only, generic error catch. Whenever possible, the most likely exception or the scenario should be specifically caught. A general exception can then be added as backup so that the program does not have to crash.

Listing 3 shows an example of catching a specific error case, followed by the general Exception case. When creating the dictionary row in the inventory list, the first entry is converted to an integer. This creates the possibility of a `ValueError` , caught on line 161, where the parameter value cannot be converted to an integer. There is also the possibility that the entry parameter did not contain all three values or was not passed as a tuple. This error is caught on line 164with the `IndexError`. After creating the dictionary row, it is then appended to the table parameter. If the `table` parameter is not passed properly as a list, the `append()` method will fail with an AttributeError. I then add the general Exception case on line 168 to catch any other unexpected issues.

```
39.     def add_CD(tplEntry, table):
40.         """Function to add a CD dictionary entry to the table/inventory
41. …

55.         """
56.         # convert tuple values into dictionary item
57.
58.         try:
59.             dicRow = {'ID': int(tplEntry[0]),
60.                       'Title': tplEntry[1],
61.                       'Artist': tplEntry[2]}
62.             table.append(dicRow)
63.             return True
64.         except ValueError:
65.             print('\n! Unable to add new CD to table. ID is not an integer.\n')
66.             return False
67.         except IndexError:
```

[12] Dawson, Michael, Python Programming 3rd Edition, Course Technology, 2010

[13] Python.org, https://docs.python.org/3/tutorial/errors.html, August 23, 2020

```
            print('\nUnable to add new CD to table. Did not have the correct' +
                  ' number of data fields for new CD entry.\n')
            return False
        except AttributeError:
            print('\n! Unable to add new CD to table. Could not append to table.\n')
        except Exception as e:
            print('Unable to add new CD to table.', e,'\n')
            return False
```

Listing 3: CDInventory.py code showing specific and general exception cases

If the try block contains a number of different code lines, the possibilities for errors may increase. You can add except blocks as needed, although only the first exception generated will be executed. Once the `try` block reaches an error, the following lines are not executed.[14]

Custom Exceptions

Python also supports custom exceptions. These can be created as their own defined classes or raised in place using the `raise` command. While presented in the Assignment 7 materials, I did not use this option in the CDInventory.py script for Assignment 7.[15]

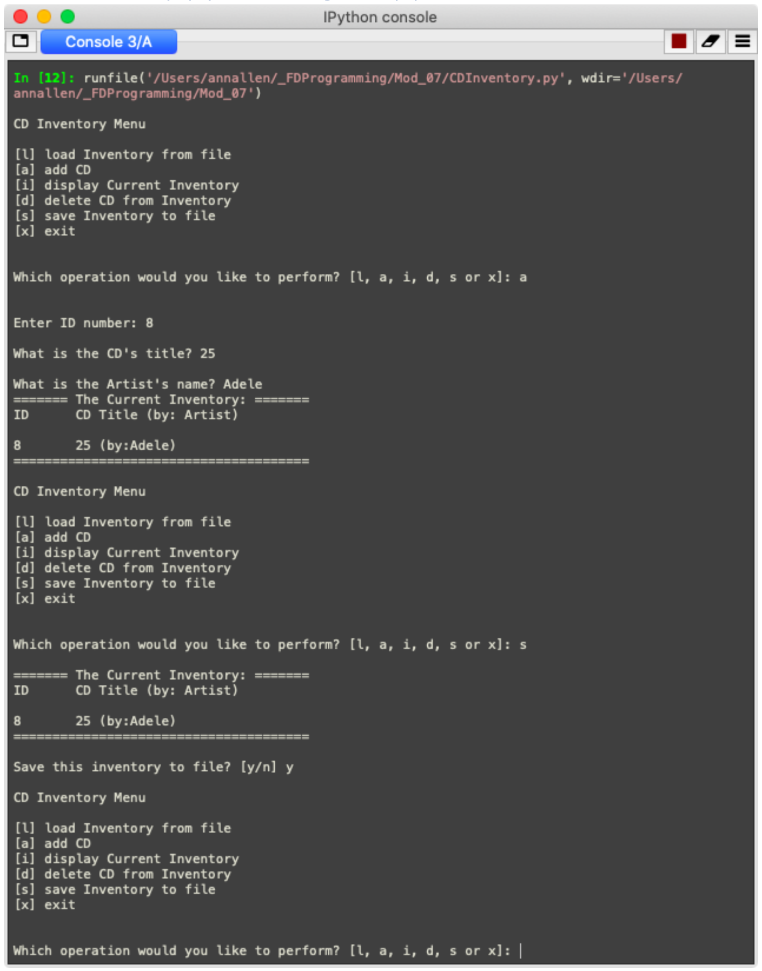
Researching Exception Handling

As I searched for more information on exception handling, I found that many of the links on the first page of google results were very similar. Most sites seemed to describe exceptions at more of an implementation level, rather than introducing a new programmer to exception handling. I might expect this at the python.org site, but I was hoping for more at PythonTutorials. I ended up checking some videos and found Corey Schafer's video, https://www.youtube.com/watch?v=NIWwJbo-9_8. This seemed more appropriate for my purposes, although his recommendation to separate the file access in its own try-block was not confirmed anywhere else. Our course textbook seems to present the material the best for someone not learning python as a second coding language. AskPython turned out to be a nice resource and even added in information about catching pickling errors, which was particularly appropriate for this Assignment.

[14] Python.org, https://docs.python.org/3/tutorial/errors.html, August 23, 2020
[15] Biesinger, Dirk, Introduction to Programming – Python, FDN_Py_Module_07.pdf, August, 23, 2020

Demonstration of working script

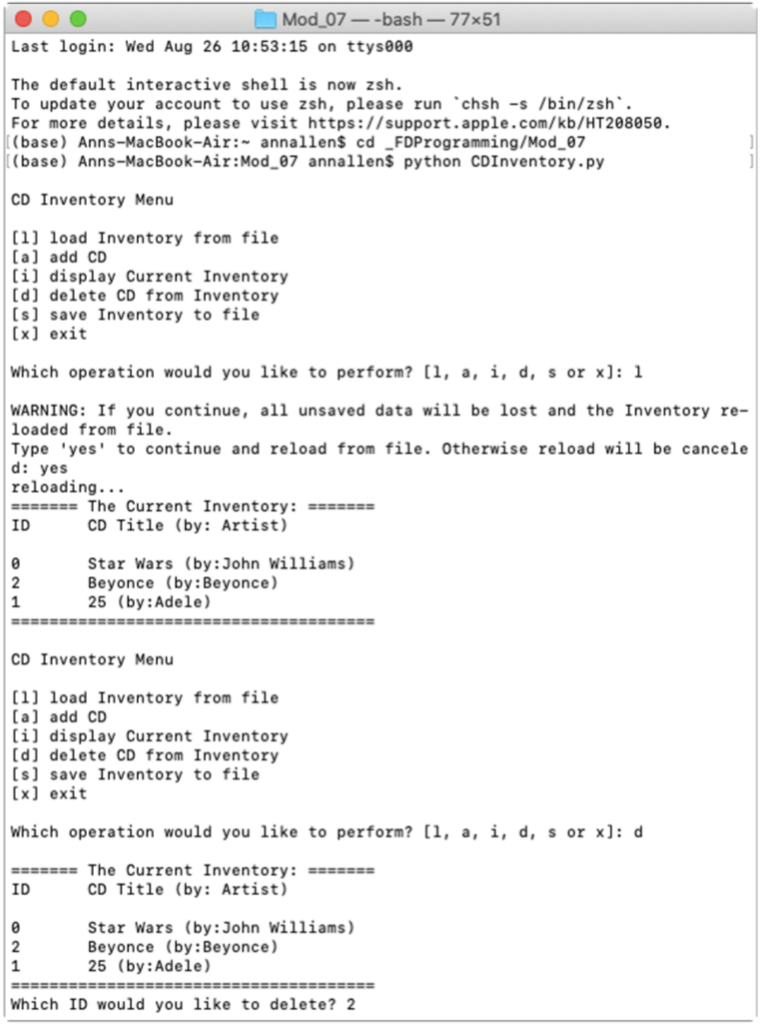
CDInventory.py running in Spyder IDE



```
In [12]: runfile('/Users/annallen/_FDProgramming/Mod_07/CDInventory.py', wdir='/Users/
annallen/_FDProgramming/Mod_07')

CD Inventory Menu

[l] load Inventory from file
[a] add CD
[i] display Current Inventory
[d] delete CD from Inventory
[s] save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: a


Enter ID number: 8

What is the CD's title? 25

What is the Artist's name? Adele
======= The Current Inventory: =======
ID      CD Title (by: Artist)

8       25 (by:Adele)
======================================

CD Inventory Menu

[l] load Inventory from file
[a] add CD
[i] display Current Inventory
[d] delete CD from Inventory
[s] save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: s

======= The Current Inventory: =======
ID      CD Title (by: Artist)

8       25 (by:Adele)
======================================

Save this inventory to file? [y/n] y

CD Inventory Menu

[l] load Inventory from file
[a] add CD
[i] display Current Inventory
[d] delete CD from Inventory
[s] save Inventory to file
[x] exit


Which operation would you like to perform? [l, a, i, d, s or x]: |
```

Figure 3: CDInventory.py running in Spyder IDE

CDInventory.py running in MacOS Bash terminal



Mod_07 — -bash — 77×51

```
Last login: Wed Aug 26 10:53:15 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) Anns-MacBook-Air:~ annallen$ cd _FDProgramming/Mod_07
(base) Anns-MacBook-Air:Mod_07 annallen$ python CDInventory.py

CD Inventory Menu

[l] load Inventory from file
[a] add CD
[i] display Current Inventory
[d] delete CD from Inventory
[s] save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: l

WARNING: If you continue, all unsaved data will be lost and the Inventory re-
loaded from file.
Type 'yes' to continue and reload from file. Otherwise reload will be cancele
d: yes
reloading...
======= The Current Inventory: =======
ID      CD Title (by: Artist)

0       Star Wars (by:John Williams)
2       Beyonce (by:Beyonce)
1       25 (by:Adele)
======================================

CD Inventory Menu

[l] load Inventory from file
[a] add CD
[i] display Current Inventory
[d] delete CD from Inventory
[s] save Inventory to file
[x] exit

Which operation would you like to perform? [l, a, i, d, s or x]: d

======= The Current Inventory: =======
ID      CD Title (by: Artist)

0       Star Wars (by:John Williams)
2       Beyonce (by:Beyonce)
1       25 (by:Adele)
======================================
Which ID would you like to delete? 2
```

Figure 4: CDInventory.py running in MacOS Bash Terminal

Summary

Using Pickling/Unpickling for saving data to and from a file in this assignment was surprisingly simple and straightforward. With all the program data stored in one object, this object can be written and read directly from the file with all the structure, including dictionary keys, intact. This was fast and efficient in one line of code. Addressing error handling, on the other hand, was more complex and less defined. I often struggled to decide where to add the error handling and how many lines to combine into each check. I also saw two or three compact lines of code turn into 40 to catch the error cases. I wished for an IDE feature to hide except blocks entirely. While the code has grown significantly in Assignment 7, with pickling and the error handling, the program feels more robust.

Appendix

CDInventory.py script

```
#------------------------------------------#
# Title: Assignment06_Starter.py
# Desc: Working with classes and functions.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# AAllen, 2020-Aug-15, Added function get_new_entry() in IO
# AAllen, 2020-Aug-15, Added functions add_CD(), delete_CD() in DataProcessor
# AAllen, 2020-Aug-15, Added function write_file() in FileProcessor
# AAllen, 2020-Aug-15, updated function comments, added get_sorted_values_list()
# AAllen, 2020-Aug-15, added error checking in get_new_entry() for ID
# AAllen, 2020-Aug-17, cleaned up some docstrings
# AAllen, 2020-Aug-18, moved call to DataProcessor.get_sorted_values_list()
#                       out of IO.get_new_entry()
# AAllen, 2020-Aug-20, removed pass calls
# AAllen, 2020-Aug-22, replaced code to in get_new_entry() to find the next
#                       available ID
# AAllen, 2020-Aug-22, updated FileProcessor.read_file() and write_file()
#                       to use a pickled binary data format
# AAllen, 2020-Aug-22, added structured error handling for IO.get_new_entry()
# AAllen, 2020-Aug-22, create IO.get_ID() to get valid ID to delete
# AAllen, 2020-Aug-23, added try-except blocks throughout code
# AAllen, 2020-Aug-26, removed try-except around simple input lines
# AAllen, 2020-Aug-26, added general Exception case catches to try-except blocks
#------------------------------------------#

# -- DATA -- #
import pickle

strChoice = '' # User input
lstTbl = []  # list of lists to hold data
dicRow = {}  # list of data row
strFileName = 'CDInventory.dat'  # data storage file
objFile = None  # file object


# -- PROCESSING -- #
class DataProcessor:
    @staticmethod
    def add_CD(tplEntry, table):
        """Function to add a CD dictionary entry to the table/inventory

        Takes a tuple of entry strings (ID, Title, Artist) and creates a
        dictionary collection {'ID':num, 'Title':string, 'Artist':string}
        that is added to the full inventory table, which is a list of dictionaries

        Args:
            tplEntry: a tuple of strings containing values to be added into
            dictionary collection and table - expected order of ID, Title, Artist.
            table (list of dict): 2D data structure (list of dicts) that holds
            the data during runtime

        Returns:
            Boolean: True if row could be added, False if it could not.

        """
        # convert tuple values into dictionary item
```

```python

        try:
            dicRow = {'ID': int(tplEntry[0]),
                      'Title': tplEntry[1],
                      'Artist': tplEntry[2]}
            table.append(dicRow)
            return True
        except ValueError:
            print('\n! Unable to add new CD to table. ID is not an integer.\n')
            return False
        except IndexError:
            print('\nUnable to add new CD to table. Did not have the correct' +
                  ' number of data fields for new CD entry.\n')
            return False
        except AttributeError:
            print('\n! Unable to add new CD to table. Could not append to table.\n')
        except Exception as e:
            print('Unable to add new CD to table.', e,'\n')
            return False


    @staticmethod
    def delete_CD(intDelID, table):
        """Function to delete a CD dictionary entry from the table/inventory

        Deletes the row identified by the ID value from the inventory table.
        The row is a dictionary entry that represents each CD.

        Args:
            intDelID: ID value for the dictionary collection in the list that
            will be deleted
            table (list of dict): 2D data structure (list of dicts) that holds
            the CD inventory data during runtime

        Returns:
            Boolean: True if row was deleted. False if not.

        """
        intRowNr = -1
        blnCDRemoved = False
        for row in table:
            intRowNr += 1
            try:
                    # if value of 'ID' for this row matches ID to delete, delete
                    # row and set flag to true that row is deleted
                    if row['ID'] == intDelID:
                        del table[intRowNr]
                        blnCDRemoved = True
                        break
                except KeyError:
                    print('Inventory table does not have expected key value.\n')
                except IndexError:
                    print('Error deleting expected row from inventory. Index out of range.\
n')
                except Exception as e:
                    print('Error deleting row from inventory.', e, '\n')

            if blnCDRemoved:
                print('The CD was removed\n')
                return True
            else:
                    print('! Unable to delete CD. Could not find CD with ID = ',
                          intDelID, '.\n')
                    return False



            @staticmethod
            def get_sorted_values_list(table, key='ID'):
                """Function to get all values of the specified key from a list of
                dictionaries

                Creates a sorted list of all the values for the key provided that are
                currently in the inventory table (2D list of dictionaries)

                Args:
                    table: expects a 2D table that is a list of dictionaries
                    key: key for which values are retrieved from the dictionaries,
                    defaults to look for a key name 'ID'

                Returns:
                    List: sorted list of all values with the provided key that are
                    in the 2D table. List is set to None if unable to get list.
                    False: if unable to retrieve list of IDs

                """
                # start with an empty list of IDs
                lstIDsUsed = []

                # if the inventory table is empty, return an empty list of IDs as
                # valid return value
                try:
                    if len(table) == 0:
                        return lstIDsUsed
                except TypeError:
                    print("Error getting length of Inventory table. Table set to none.\n")
                    return False

                # if the inventory table is not empty get current IDs
                try:
                    #get a list of all values for the key parameter
                    for row in table:
                        lstIDsUsed.append(row.get(key))
                    lstIDsUsed.sort()
                except TypeError:
                    # if lstIDsUsed is still empty after getting IDs, .sort() will
                    # cause a TypeError. Catch this error and return False so
                    # duplicate IDs are not created
                    print('TypeError getting list of IDs from table.\n')
                    lstIDsUsed = False
                    return lstIDsUsed
                except Exception as e:
                        print('Error getting list of IDs from table.', e, '\n')
                        lstIDsUsed = False
                else:
                    return lstIDsUsed



        class FileProcessor:
            """Processing the data to and from text file"""

        @staticmethod
        def read_file(file_name, table):
            """Function to manage data ingestion from file to a list of dictionaries

            Reads the data from file identified by file_name into a 2D table
            (list of dicts) table one line in the file represents one
            dictionary row in table.

            Args:
                file_name (string): name of file used to read the data from
                table (list of dict): 2D data structure (list of dicts) that holds
                the data during runtime

            Returns:
                Boolean: True if file loaded. False if not loaded.
            """
            # clear table to load with file contents
            table.clear()
            try:
                # open file to read binary
                objFile = open(file_name, 'rb')
            except FileNotFoundError as e:
                print('! FileNotFoundError reading file: ', e ,'\n')
                return False
            except PermissionError as e:
                print('! PermissionError reading file: ', e ,'\n')
                return False
            except Exception as e:
                print('! Error reading file: ', e ,'\n')
                return False
            else:
                try:
                    # copy contents of file into table
                    table[:] = pickle.load(objFile)[:]
                except AttributeError as e:
                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
                          'Recommended to check data file for corruption.')
                    return False
                except pickle.UnpicklingError as e:
                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
                          'Recommended to check data file for corruption.')
                    return False
                except EOFError as e:
                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
                          'Recommended to check data file for corruption.')
                    return False
                except Exception as e:
                    print('! Error unpickling data file: ', file_name,', ', e ,'\n',
                          'Recommended to check data file for corruption.')
                    return False
                objFile.close()
                return True



        @staticmethod
        def write_file(file_name, table):
            """Function to write updated inventory list of dictionaries to file

            Writes the data in table to the file identified by file_name. Table is
            a list of dictionaries. Each dictionary item is written to a line in
```

```python
            the file, with a newline ending. The dictionary is written as the
            value from each key/value pair separated by a comma

            Args:
                file_name (string): name of file used to save to
                table (list of dict): 2D data structure (list of dicts) that holds
                the data during runtime

            Returns:
                Boolean: True if file could be updated. False if file could not
                be updated.
            """

            try:
                # open file to write binary
                objFile = open(file_name, 'wb')
            except PermissionError as e:
                print('! PermissionError reading file: ', e ,'\n')
                return False
            except Exception as e:
                print('! Error reading file: ', e ,'\n')
                return False
            else:
                try:
                    # add table data to file
                    pickle.dump(table, objFile)
                except AttributeError as e:
                    print('! Error pickling data file: ', e, '\n')
                    return False
                except pickle.PicklingError as e:
                    print('! Error pickling data file: ', e ,'\n')
                    return False
                except Exception as e:
                    print('! Error pickling data file: ', e ,'\n')
                    return False
                objFile.close()
                return True


    # -- PRESENTATION (Input/Output) -- #

    class IO:
        """Handling Input / Output"""

        @staticmethod
        def print_menu():
            """Displays a menu of choices to the user

            Args:
                None.

            Returns:
                None.
            """

            print('\nCD Inventory Menu\n\n[l] load Inventory from file\n[a] add CD\n[i] dis
play Current Inventory')
            print('[d] delete CD from Inventory\n[s] save Inventory to file\n[x] exit\n')


        @staticmethod
    def menu_choice():
        """Gets user input for menu selection

        Args:
            None.

        Returns:
            choice (string): a lower case sting of the users input out of the choices l, a, i, d, s or x

        """

        choice = ' '
        try:
            # request user menu choice and check for valid option
            while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
                choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
        except:
            print('Error getting new menu option.\n')

        print()  # Add extra space for layout
        return choice


    @staticmethod
    def show_inventory(table):
        """Displays current inventory table


        Args:
            table (list of dict): 2D data structure (list of dicts) that
            holds the data during runtime.

        Returns:
            None.

        """
        try:
            print('======= The Current Inventory: =======')
            print('ID\tCD Title (by: Artist)\n')
            for row in table:
                print('{}\t{} (by:{})'.format(*row.values()))
            print('======================================')
        except:
            print("Error displaying Inventory menu.\n")


    @staticmethod
    def get_new_entry(lstTestIDs):
        """Requests the new entry values, ID, Title, Artist

         Args:
            lstTestIDs: a list containing IDs currently in use in the table

        Returns:
            tplNewEntry: a tuple of three strings with the user's input for the
            (ID, Title, Artist) for a new CD entry

        """
        # Get a unique entry ID
```

```python
        entryID = '' # create empty default ID
        validID = False # Flag to indicate valid ID identified
        # request an ID number from user
        entryID = input('Enter ID number: ').strip()

        # verify ID number and re-ask as needed
        while not validID:
            try:
                int(entryID)
            except ValueError:
                # if the user does not enter an ID, find one and assign it
                # if the list of current IDs is empty, start with ID 0
                if entryID == '' and ((lstTestIDs == []) or (lstTestIDs == None)):
                    entryID = 0
                    break
                # assign a CD ID, by finding the next available, unique ID to use
                # code curtesy of Doug Klos
                elif entryID == '':
                    entryID = 0
                    while True:
                         if entryID in lstTestIDs:
                            entryID += 1
                         else:
                             break
                # if they ID was not a valid int and not blank, ask again
                else:
                    entryID = input('Please enter a numerical, non-decimal ID number: ').strip()
            # if an ID was passed in that was not a string or hits another error, try again
            except:
                entryID = input('Unable to assign ID.\n' +
                                'Please choose a different ID or press [enter] to' +
                                ' assign one automatically: ').strip()
            else:
                # if entry is a valid int and not a duplicate, use the ID
                if  int(entryID) in lstTestIDs:
                    entryID = input('ID =' + entryID + ' is already being used.\n' +
                                     'Please choose a different ID or [enter] to' +
                                     ' assign one automatically: ').strip()
                elif int(entryID) < 0:
                    entryID = input('Please choose a positive value ID or [enter] to' +
                                     ' assign one automatically: ').strip()
                else:
                    validID = True

        # get the entry title and artist, these can be blank

        entryTitle = input('What is the CD\'s title? ').strip()
        entryArtist = input('What is the Artist\'s name? ').strip()

        # create new entry tuple with ID, Title and Artist
        tplNewEntry = (entryID, entryTitle, entryArtist)
        return tplNewEntry


    @staticmethod
    def get_ID():
        """Requests a positive, integer ID value from the user to delete

```

```python
         Args:
            none

        Returns:
            Integer: integer value greater than 0 for ID to delete

        """
        while True:
            try:
                # request ID from user
                intIDDel = int(input('Which ID would you like to delete? ').strip())
            except ValueError:
                # catch alpha/symbol error entries that do not convert to int
                print('\nPlease enter an integer ID value greater than zero.')
            else:
                # Check to see if the integer is a positive value
                if intIDDel >= 0:
                    break
                else:
                    print('\nPlease enter an integer ID greater than 0.')
        return intIDDel


# 1. When program starts, read in the currently saved Inventory
FileProcessor.read_file(strFileName, lstTbl)

# 2. start main loop
while True:
    # 2.1 Display Menu to user and get choice
    IO.print_menu()
    strChoice = IO.menu_choice()
    # 3. Process menu selection
    # 3.1 process exit first
    if strChoice == 'x':
        break
    # 3.2 process load inventory
    if strChoice == 'l':
        print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
        strYesNo = input('Type \'yes\' to continue and reload from file. Otherwise reload will be canceled: ')
        try:
            if strYesNo.lower() == 'yes':
                print('reloading...')
                FileProcessor.read_file(strFileName, lstTbl)
            else:
                input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
        except:
            print('Error loading data from file.\n')
        else:
            IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.3 process add a CD
    elif strChoice == 'a':
        # 3.3.0 get the sorted list of IDs currently in use
        lstTestIDs = DataProcessor.get_sorted_values_list(lstTbl)
        # 3.3.1 Check to see if there is a valid list of IDs to check against
        if lstTestIDs == False:
            print('Unable to add a new entry without a confirming current IDs in use.\n')
            # 3.3.2 Ask user for new ID, CD Title and Artist
            else:
                #3.3.2.1 Get values for the new entry
                tplUserEntry = IO.get_new_entry(lstTestIDs)
                # 3.3.2.2 Add item with users values to the table
                DataProcessor.add_CD(tplUserEntry, lstTbl)
            # 3.3.3 Display inventory to user to confirm CD added
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.4 process display current inventory
        elif strChoice == 'i':
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.5 process delete a CD
        elif strChoice == 'd':
            # 3.5.1 get Userinput for which CD to delete
            # 3.5.1.1 display Inventory to user
            IO.show_inventory(lstTbl)
            # 3.5.1.2 ask user which ID to remove
            intIDDel = IO.get_ID()
            # 3.5.2 search thru table and delete CD
            # 3.5.2.1 delete CD with requested ID
            DataProcessor.delete_CD(intIDDel, lstTbl)
            # 3.5.2.2 display inventory to user after delete for confirmation
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.6 process save inventory to file
        elif strChoice == 's':
            # 3.6.1 Display current inventory and ask user for confirmation to save
            IO.show_inventory(lstTbl)
            strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
            # 3.6.2 Process choice
            if strYesNo == 'y':
                # 3.6.2.1 save data
                FileProcessor.write_file(strFileName, lstTbl)
            else:
                input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
                print('Error confirming save cancelation with user.\n')
            continue  # start loop back at top.
        # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be save:
        else:
            print('General Error')
```