Alex Angelico
August 27, 2020
Foundations of Programming (Python)
CD Inventory with Binary Storage & Structured Error Handling

Binary Data Storage and Structured Error Handling

Introduction

This assignment will illustrate the value of implementing two common but more advanced concepts in Python and other programming languages—binary data storage and structured error handling—in the context of the CD Inventory program. Specific functionalities and constructs discussed will include pickling, try-except, and exception classes. The assignment will also briefly introduce Markdown language.

Binary File Storage vs Text File Storage

Although it is most obvious and, in some ways, easiest to store data used and generated by a program in the form of a text file, it is too simple a solution for most practical purposes. This is because text files are, limited to storing plain text characters; in binary files however, you can store more complex data very efficiently.[1] In Python, this is accomplished using a module called “pickle”, and it is useful for storing objects like lists and dictionaries, among other things.[2]

```
    @staticmethod
    def read_file(file_name, table):
```

…

```
                with open(file_name, 'rb') as fileObj:
                    data = pickle.load(fileObj)
```

Listing 1 - Excerpt, Appendix A: Core binary file reading functionality

```
    @staticmethod
    def write_file(file_name, table):
```

…

```
                with open(file_name, 'wb') as objFile:
                    pickle.dump(table, objFile)
```

Listing 2 - Excerpt, Appendix A: Core binary file writing functionality

The code in Listings 1 and 2 illustrates that, aside from being an efficient method for storing complex data, binary file access instructions using the pickle module are also very economical to write in Python compared to text file access. This is because functional code for binary file access does not require several lines of instruction that are typically required to translate between complex data structures in a program and the plain text characters that are stored in a text file. In other words, pickling data in Python ensures it is stored in the same format that a program uses to manipulate it—the only tradeoff being that the resulting binary file is unreadable by humans on its own. [3]

[1] “Chapter 7 Files and Exceptions: The Trivia Challenge Game”, Python Programming for the Absolute Beginner Third Edition, Michael Dawson, p200.
[2] “How to Use Pickle to Save Objects in Python”, ThoughtCo., Al Lukaszewski, https://www.thoughtco.com/using-pickle-to-save-objects-2813661 (accessed August 27 2020).
[3] “Module 07 of 10”, *Foundations of Programming (Python),* Dirk Biesinger, p14 (accessed August 27, 2020).

On that note, pickled data is not only unreadable by humans. The transformation protocol that pickle uses to read and write binary file data is specific to Python, meaning there is no guarantee it will allow for file compatibility across programs written in other languages—nor even with differing versions of Python.[4] With that in mind, pickling should only be relied on for use within a controlled Python ecosystem.

Structured Error Handling

Questions of compatibility are inherent within the operation of an individual program as well. Whether it be file access or data manipulation or user IO, there are many points in a program where the introduction of something unexpected, even a small data discrepancy, can trigger an error that causes it to terminate in an uncontrolled way.

```
38.     @staticmethod
39.     def delete_cd(delID, table):
```

…

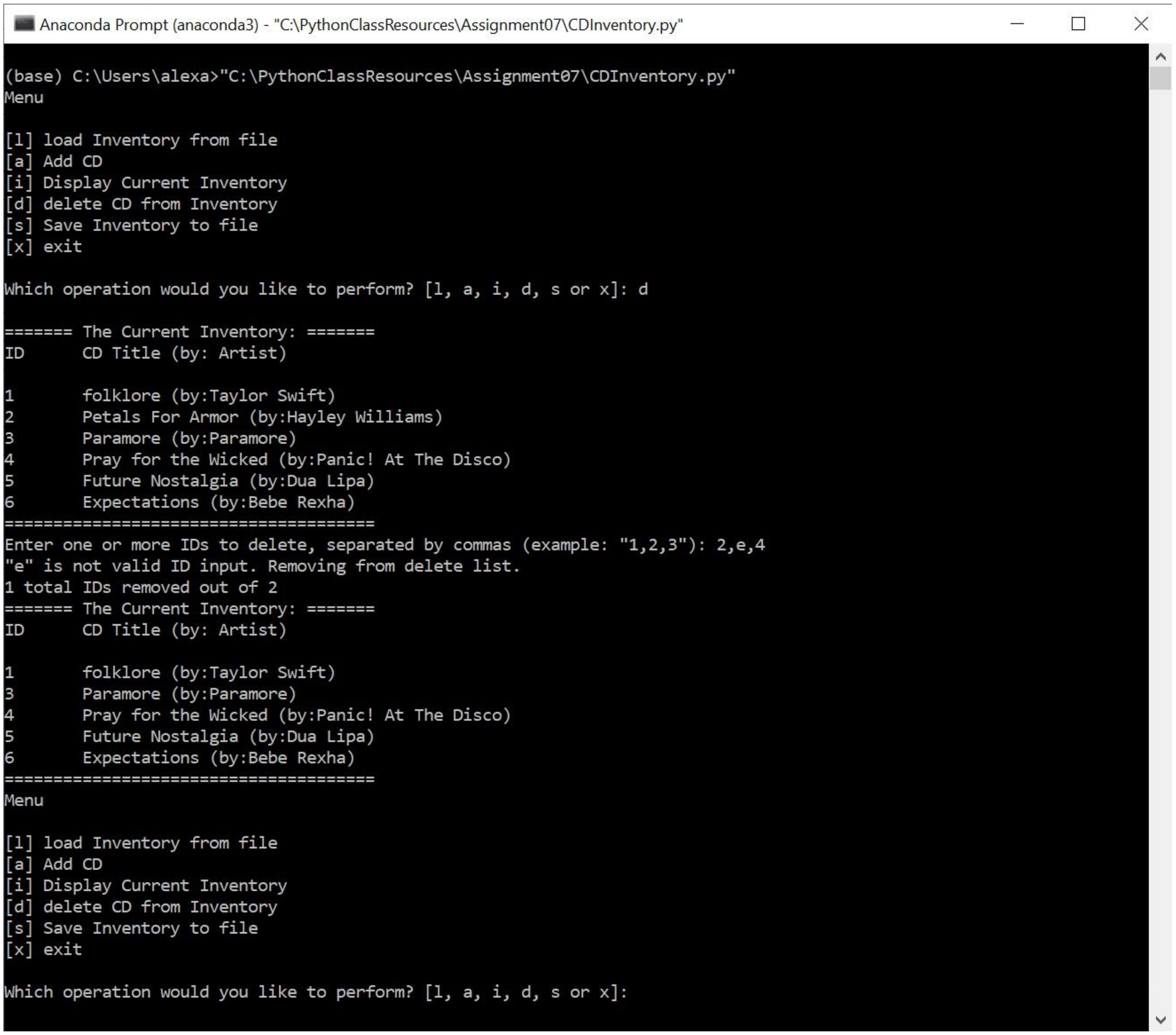
```
49.         remove_count = 0
50.         for row in delID: delID = row.strip().split(',')
51.
52.         for _id in delID:
53.             try:
54.                 _id = int(_id)
55.             except:
56.                 print('"''"',_id,'"',' is not valid ID input. Removing from delete list.', sep='')
57.                 delID.remove(_id)
58.             pass
59.             for row in table:
60.                 if row['ID'] == _id:
61.                     remove_count += 1
62.                     table.remove(row)
63.
64.         if remove_count == 0: print('No matching IDs')
65.         else: print(f'{remove_count} total IDs removed out of {len(delID)}')
```

Listing 3 - Excerpt, Appendix A: Structured error handling in the delete_cd() function

The way that the conversion of variable _id from string to integer type data on line 54 is performed in the delete_cd() function in Listing 3 highlights how structured error handling can be used to preserve the execution instance of a the CD Inventory program where an uncontrolled error would otherwise result. This is done by isolating code that is vulnerable to something going wrong (in this case, bad input provided by a human user) within a try-except construct, which allows for a programmer-designed custom response to erroneous conditions. [5]

In this case, the particular error type that is being guarded against is a value error. The specific way in which a value error might be introduced in this function is via user input passed as an argument into the delID parameter. For instance, if the user, at the point in the program where they are prompted to input numerical ID values to delete from the CD Inventory, instead provided alphabetical input, the instruction on line 54 would fail to execute when the program enters the try statement—and if there were further lines contained within the try statement, the program would not even proceed to execute them.[6] Because alphabetical characters cannot be converted to the integer data type, this would cause the error—but the problem is instead handled by the except statement, which instructs the program to perform the operations of printing out a custom error message, which contains information about what went wrong and how it is being handled. Following that, the program continues to run.

[4] "Pickle in Python: Object Serialization", DataCamp, Theo Vanderheyden, https://www.datacamp.com/community/tutorials/pickle-python-tutorial (accessed August 27 2020).
[5] "Module 07 of 10", Dirk Biesinger, p16-17 (accessed August 27, 2020).
[6] "Exception Handling in Python", PythonForBeginners, https://www.pythonforbeginners.com/error-handling/exception-handling-in-python (accessed August 27 2020).



Anaconda Prompt (anaconda3) - "C:\PythonClassResources\Assignment07\CDInventory.py"

```
(base) C:\Users\alexa>"C:\PythonClassResources\Assignment07\CDInventory.py"
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)

1       folklore (by:Taylor Swift)
2       Petals For Armor (by:Hayley Williams)
3       Paramore (by:Paramore)
4       Pray for the Wicked (by:Panic! At The Disco)
5       Future Nostalgia (by:Dua Lipa)
6       Expectations (by:Bebe Rexha)
======================================
Enter one or more IDs to delete, separated by commas (example: "1,2,3"): 2,e,4
"e" is not valid ID input. Removing from delete list.
1 total IDs removed out of 2
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       folklore (by:Taylor Swift)
3       Paramore (by:Paramore)
4       Pray for the Wicked (by:Panic! At The Disco)
5       Future Nostalgia (by:Dua Lipa)
6       Expectations (by:Bebe Rexha)
======================================
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 1 - Structured error handling while attempting to delete CDs with bad input

In Figure 1, notice how even though bad erroneous alphabetical input is provided by the user while attempting to delete CDs, the relevant program function is still able to execute normally and return the user to the main menu.

Sidebar: The Exception Class

The try-except construct can also be implemented in such a way that it provides further context for an error extracted from the exception class. This is because Python uses an inbuilt exception class hold information about an error, which is generated as an object whenever a program actually causes an error.[7]

```
71.     @staticmethod
72.     def read_file(file_name, table):
```

…

```
85.         table.clear()
```

[7] Ibid, p18 (accessed August 27, 2020).

```
86.
87.         readcount = 0
88.         while readcount < 1:
89.             try:
90.                 with open(file_name, 'rb') as fileObj:
91.                     data = pickle.load(fileObj)
92.                     readcount += 1
93.             except(IOError):
94.                 print('CD Inventory file not found. Terminating program.')
95.                 sys.exit()
96.
97.         for line in data:
98.             try:
99.                 if line != {}: table.append(line)
100.                   except(TypeError, ValueError):
101.                       print('Corrupt or missing data.')
102.                       sys.exit()
```

Listing 4 - Multiple types of exceptions contained within the exception class, triggering different custom error handling messages

Listing 4 provides a sample of different exceptions within the exception class (lines 93 and 100) which are produced by different errors that might result from the execution of the read_file() function.

However, it is also possible to derive entirely new subsidiary exception classes by using the same class structure that is used to create functions. The only difference is that the defined name for the class must be followed by "(Exception)", and then the programmer-defined output the exception generates following below.[8] The reason a programmer might elect to do this is that they can develop structured error handling for the specific needs of their code that isn't covered by Python's pre-existing exception classes.[9]

Markdown Language

Markdown language is a collection of conventions for writing structured documents in plain text formats. It is intended to provide a high degree of organizational clarity, and in the context of programming, can often be rendered in HTML and other formats.

Summary

In some ways, this assignment was especially difficult for me to complete. While learning how to implement binary file access was not challenging, I am still coming to terms with how to effectively implement structured error handling through the try-except construct. I will continue to work on it in the coming weeks.

Appendix

A – CD Inventory Code

```
# -- DATA -- #
menuChoice = '' # 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
```

[8] Ibid, p22-23 (accessed August 27, 2020).
[9] "Exception Handling in Python", OverIQ.com, https://overiq.com/python-101/exception-handling-in-python/ (accessed August 27 2020).

```python
    def add_cd(ID, title, artist, table):
        """Collects new CD data from user and converts into dict for appending to current inventory table

        Args:
            ID (string): numerical identification for the new CD
            title (string): album title of the new CD
            artist (string): artist name of the new CD
            table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime

        Returns:
            None.
        """
        dicRow = {'ID': ID, 'Title': title, 'Artist': artist}
        table.append(dicRow)
        print()

    @staticmethod
    def delete_cd(delID, table):
        """Deletes dicts by ID key from 2D data structure

        Args:
            delID (list of strings): holds one or more ID values designated for deletiton
            table (list of dicts): 2D data structure (list of dicts) that holds the data during runtime

        Returns:
            None.
        """
        remove_count = 0
        for row in delID: delID = row.strip().split(',')

        for _id in delID: # I do not know if it is possible complete a for loop after a nested except condition is triggered, but if it is I don't know how
            try:
                _id = int(_id)
            except:
                print('"''"',_id,'"',' is not valid ID input. Removing from delete list.', sep='')
                delID.remove(_id)
            pass
            for row in table:
                if row['ID'] == _id:
                    remove_count += 1
                    table.remove(row)

        if remove_count == 0: print('No matching IDs')
        else: print(f'{remove_count} total IDs removed out of {len(delID)}')

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:
            None.
        """
        table.clear()
```

```

 readcount = 0
 while readcount < 1:
     try:
         with open(file_name, 'rb') as fileObj:
             data = pickle.load(fileObj)
             readcount += 1
     except(IOError):
         print('CD Inventory file not found. Terminating program.')
         sys.exit()

 for line in data:
     try:
         if line != {}: table.append(line)
     except(TypeError, ValueError):
                print('Corrupt or missing data.')
                sys.exit()

    @staticmethod
    def write_file(file_name, table):
        """Function to manage transcription of data from list of dictionaries in
        current inventory to file

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

        Returns:
            None.
        """
        tempTbl = []
        FileProcessor.read_file(strFileName, tempTbl)
        savecount = 0
        while savecount < 1:
            try:
                if str(table) != str(tempTbl):
                    with open(file_name, 'wb') as objFile:
                        pickle.dump(table, objFile)
            except:
                print('There are no new CDs in Inventory to save.')
            savecount += 1

# -- 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('Menu\n\n[l] load Inventory from file\n[a] Add CD\n[i] Display 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:
```

```python
            None.

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

        """
        options = ['l', 'a', 'i', 'd', 's', 'x']
        while True:
            choice = input('Which operation would you like to perform? [l, a, i, d, s or x]: ').lower().strip()
            try:
                if options.count(choice) == 1: break
            except:
                continue
        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.

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

    @staticmethod
    # def add_cd_input(table, IDchecklist):
    def add_cd_input(table):
        """Gets user input for new CD information

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

        Returns:
            ID (string): numerical identification for the new CD
            title (string): album title of the new CD
            artist (string): artist name of the new CD
        """

        # We generate a list of all used ID's.
        used_ids = []
        for row in table:
            used_ids.append(row["ID"])

        # Prompt the user for an ID value
        while True:
            ID = input('Enter numerical ID: ').strip()
            try:
                ID = int(ID)
                break
            except:
                continue

        # If that ID value is currently in user, continue prompting.
        while ID in used_ids:
            print('That ID already exists. Please enter a new ID.')
            while True:
                ID = input('Enter numerical ID: ').strip()
                try:
                    ID = int(ID)
                    break
                except:
                    continue

        while True:
            title = input('What is the CD\'s title? ').strip()
            try:
                if (len(title)) > 0: break
            except:
                continue

        while True:
            artist = input('What is the Artist\'s name? ').strip()
            try:
                if (len(artist)) > 0: break
            except:
                continue

        return ID, title, artist

# 0. Import data storage and error handling modules
import pickle, sys

# 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()
    menuChoice = IO.menu_choice()

    # 3. Process menu selection
    # 3.1 process exit first
    if menuChoice == 'x':
        break
    # 3.2 process load inventory
    if menuChoice == 'l':
        print('WARNING: If you continue, all unsaved data will be lost and the Inventory re-loaded from file.')
        overwrite_verification = input('Type \'yes\' to continue and reload from file. Otherwise reload will be canceled.\n')
        if overwrite_verification.lower() == 'yes':
            print('reloading...')
            FileProcessor.read_file(strFileName, lstTbl)
            IO.show_inventory(lstTbl)
        else:
            input('canceling... Inventory data NOT reloaded. Press [ENTER] to continue to the menu.')
            IO.show_inventory(lstTbl)
        continue  # start loop back at top.
    # 3.3 process add a CD
    elif menuChoice == 'a':
        print("Please provide new CD info.")
        add_verification = 'y'
        while add_verification == 'y':
            # 3.3.1 Ask user for new ID, CD Title, and Artist
            cdID,cdTitle,cdArtist = IO.add_cd_input(lstTbl)
            # 3.3.2 Verify input data is correct
            print('\nYou entered:')
            print('ID\tCD Title (by: Artist)')
            print(f'{cdID}\t{cdTitle} (by:{cdArtist})\n')
            input_validation = input("Is this information correct? [y/n]: ").lower()
```

```python
                if input_validation == "y":
                    pass
                elif input_validation == "n":
                    continue
                # 3.3.3 Add item to the table and ask user if they want to add another CD
                DataProcessor.add_cd(cdID, cdTitle, cdArtist, lstTbl)
                while True:
                    add_verification = input('Would you like to add another CD? [y/n] ').lower()
                    try:
                        if add_verification == 'y': break
                        elif add_verification == 'n': break
                    except:
                        continue
            print()
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.4 process display current inventory
        elif menuChoice == 'i':
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.5 process delete a CD
        elif menuChoice == '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
            while True:
                cdIDdel = [input('Enter one or more IDs to delete, separated by commas (example: "1,2,3"): ').strip()]
                try:
                    if cdIDdel != ['']: break
                except:
                    continue
            # 3.5.2 search through table and delete CD
            DataProcessor.delete_cd(cdIDdel, lstTbl)
            # 3.5.3 display altered Inventory to user
            IO.show_inventory(lstTbl)
            continue  # start loop back at top.
        # 3.6 process save inventory to file
        elif menuChoice == 's':
            # 3.6.1 Display current inventory and ask user for confirmation to save
            IO.show_inventory(lstTbl)
            while True:
                save_verification = input('Save this inventory to file? [y/n] ').strip().lower()
                # 3.6.2 Process choice
                try:
                    if save_verification == 'y':
                        # 3.6.2.1 save data
                        FileProcessor.write_file(strFileName, lstTbl)
                        break
                    elif save_verification == 'n':
                        input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
                        break
                except:
                    continue
            continue  # start loop back at top.
        # 3.7 catch-all should not be possible, as user choice gets vetted in IO, but to be safe:
        else:
            print('General Error')
```

B – Github Link

https://github.com/Alex-Angelico/Assignment_07