Benjamin Reinhart

February 28, 2021

IT FDN 110 A

Assignment 07

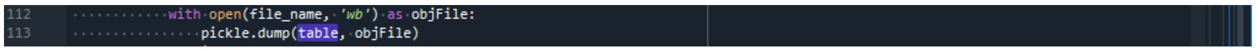
Error Handling and Binary Files

Introduction

In this assignment I will cover how to modify a script in Python that manages a CD Inventory. I will modify the script to handle a variety of errors and read in or save data using binary data instead of the original text files. We will handle errors involved with user interaction, type casting, and file access. I will use the information learned in Module 07 about error handline and binary data in order to do this. Here is a link to the GitHub repository where this is saved: https://github.com/reinhartben/Assignment_07.git

Modifying the Script

I started modifying the script by updating it to use binary data. To do this, I imported the Pickle package as we did in LAB07_B. This includes adding the letter 'b' to the file access type to signal to use binary[1]. I needed to update the two functions where we interact with files, so I updated the read_file function and the write_file function. In Figure 1 below you can see an example.



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

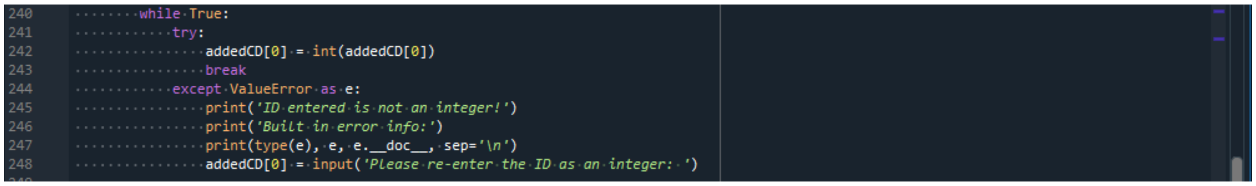
Figure 1: Working with Binary Data

We also wanted to update the script to handle a variety of potential errors. Errors often happen when we have interaction with humans, interaction with files, or when we are changing data types[2]. To do this, I worked with the built in error handling that is provided by Python and 'trapping' the code into a try-except block[3].

Going through the program I identified the locations where I think an error could occur and decided what type of error could happen. This allowed me to choose how to handle the error and catch specific exceptions. The two functions where we are interacting with the file could have errors around access to the file, the add_CD function casts a string to an integer and if the user input is incorrect, could error then, and there is always a chance of an error with the input.

[1] Dirk Biesinger, Foundations of Programming (Python), Module 07 Page 16
[2] Biesinger, Foundations of Programming (Python), Module 07 Page 17
[3] Biesinger, Foundations of Programming (Python), Module 07 Page 17

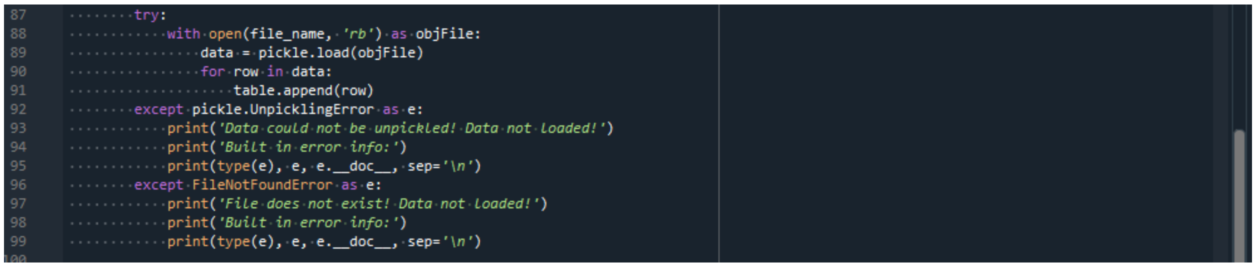
Starting with the add CD functionality, I put the functionality into a try-except block and used the ValueError when casting the string to an integer. If the ID value entered by the user cannot be converted to an integer, then we print out some error info to the user and ask for an updated ID value before continuing. You can see this in Figure 2 below.



```
        while True:
            try:
                addedCD[0] = int(addedCD[0])
                break
            except ValueError as e:
                print('ID entered is not an integer!')
                print('Built in error info:')
                print(type(e), e, e.__doc__, sep='\n')
                addedCD[0] = input('Please re-enter the ID as an integer: ')
```

Figure 2: ValueError Class

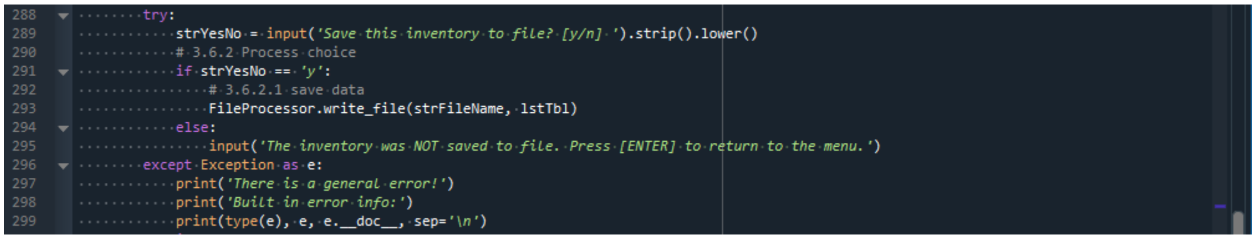
For the file interaction functions I used the FileNotFoundError class in order to capture that error[4]. I also included a catch for any issues with reading or saving the binary data so used the pickle exceptions UnpicklingError and PicklingError to capture those.



```
        try:
            with open(file_name, 'rb') as objFile:
                data = pickle.load(objFile)
                for row in data:
                    table.append(row)
        except pickle.UnpicklingError as e:
            print('Data could not be unpickled! Data not loaded!')
            print('Built in error info:')
            print(type(e), e, e.__doc__, sep='\n')
        except FileNotFoundError as e:
            print('File does not exist! Data not loaded!')
            print('Built in error info:')
            print(type(e), e, e.__doc__, sep='\n')
```

Figure 3: FileNotFoundError and UnpicklingError

The other places in the programs that take user input, I used the Exception class to catch the error and print a message to the user. This would be a more general error.

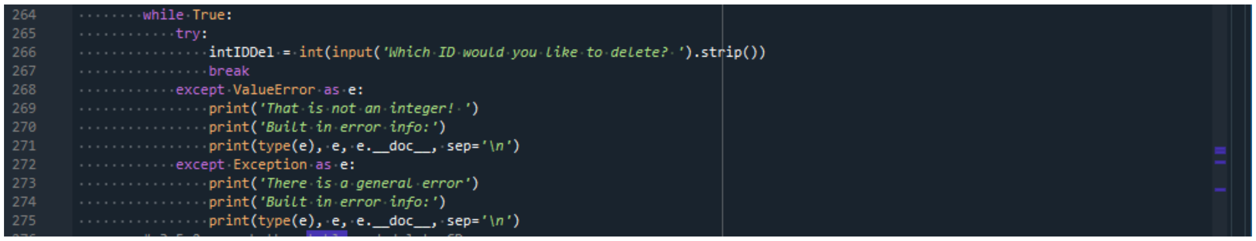


```
        try:
            strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()
            # 3.6.2 Process choice
            if strYesNo == 'y':
                # 3.6.2.1 save data
                FileProcessor.write_file(strFileName, lstTbl)
            else:
                input('The inventory was NOT saved to file. Press [ENTER] to return to the menu.')
        except Exception as e:
            print('There is a general error!')
            print('Built in error info:')
            print(type(e), e, e.__doc__, sep='\n')
```

Figure 4: Exception Class

For the delete functionality, I used both the general Exception class and the ValueError class. This was because the most likely error is the user enters something that cannot be converted to an integer, but there could be another error during input.

[4] Biesinger, Foundations of Programming (Python), Module 07 Page 17



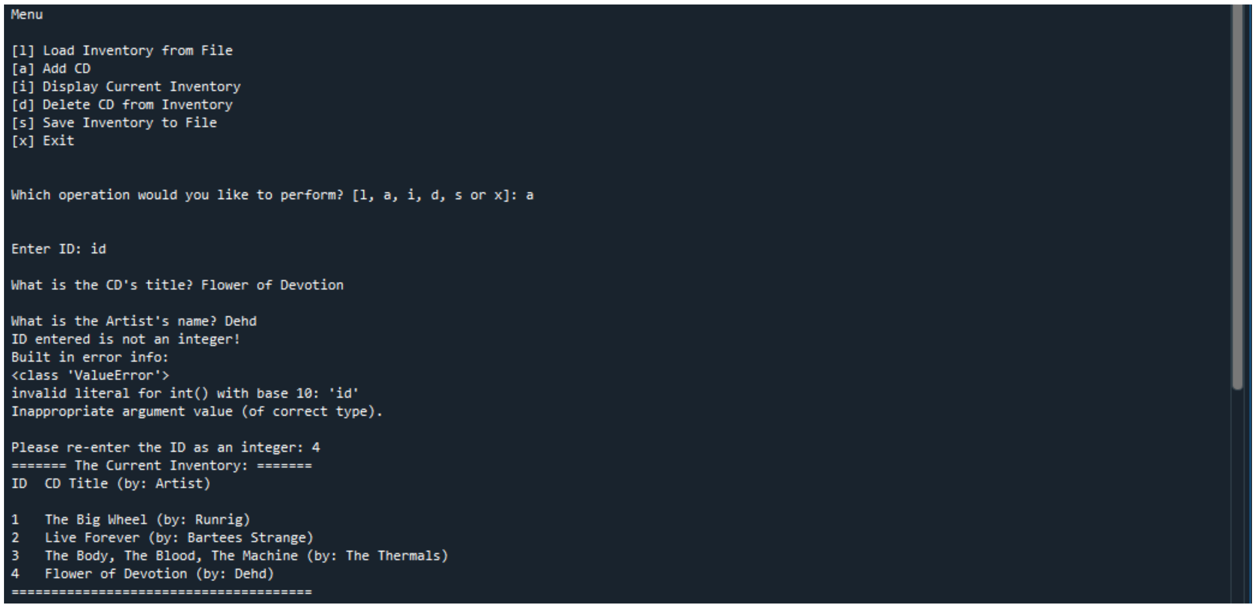
```
        while True:
            try:
                intIDDel = int(input('Which ID would you like to delete? ').strip())
                break
            except ValueError as e:
                print('That is not an integer! ')
                print('Built in error info:')
                print(type(e), e, e.__doc__, sep='\n')
            except Exception as e:
                print('There is a general error')
                print('Built in error info:')
                print(type(e), e, e.__doc__, sep='\n')
```

Figure 5: ValueError & Exception Classes

Now we should be catching any errors as they come up and handling them promptly.

Running the Script - Spyder

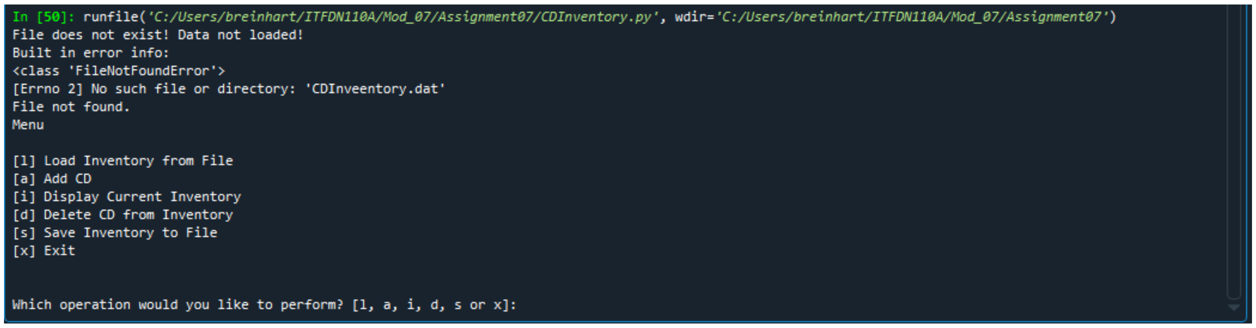
To test the modifications I made, I first ran the program in Spyder. I wanted to try to break the program and see my error handling work. First I entered a value that could not be converted to an integer below. You can see the message that is printed when the exception is thrown and then the program asks for me to input a new ID.



```
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: id

What is the CD's title? Flower of Devotion

What is the Artist's name? Dehd
ID entered is not an integer!
Built in error info:
<class 'ValueError'>
invalid literal for int() with base 10: 'id'
Inappropriate argument value (of correct type).

Please re-enter the ID as an integer: 4
======= The Current Inventory: =======
ID  CD Title (by: Artist)

1   The Big Wheel (by: Runrig)
2   Live Forever (by: Bartees Strange)
3   The Body, The Blood, The Machine (by: The Thermals)
4   Flower of Devotion (by: Dehd)
======================================
```

Figure 6: Spyder ValueError

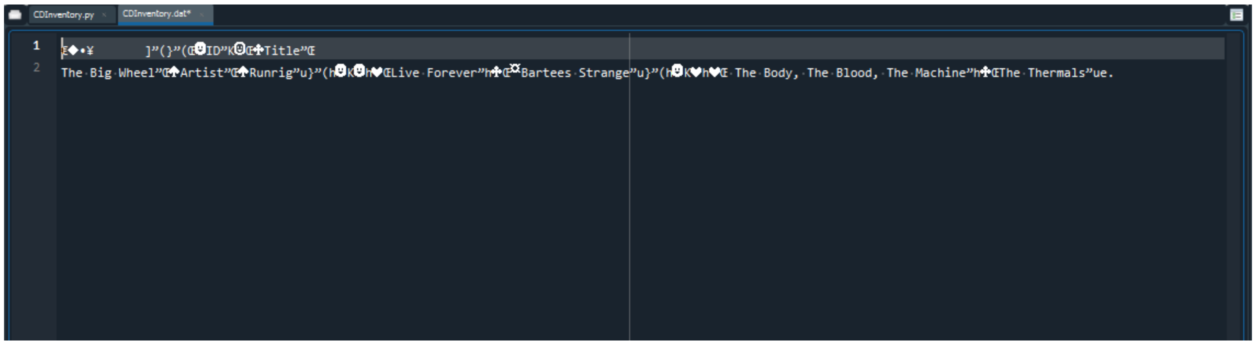
The other way to break the program is to try to use a file that does not exist. Below you can see the message noting that data was not imported into the program after I ran the program with a non-existing file.



```
In [50]: runfile('C:/Users/breinhart/ITFDN110A/Mod_07/Assignment07/CDInventory.py', wdir='C:/Users/breinhart/ITFDN110A/Mod_07/Assignment07')
File does not exist! Data not loaded!
Built in error info:
<class 'FileNotFoundError'>
[Errno 2] No such file or directory: 'CDInveentory.dat'
File not found.
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 7: Spyder FileNotFoundError

Other than that, the program still runs as intended and as it did the previous week. Below is a screenshot of the .dat file showing that it has been saved.

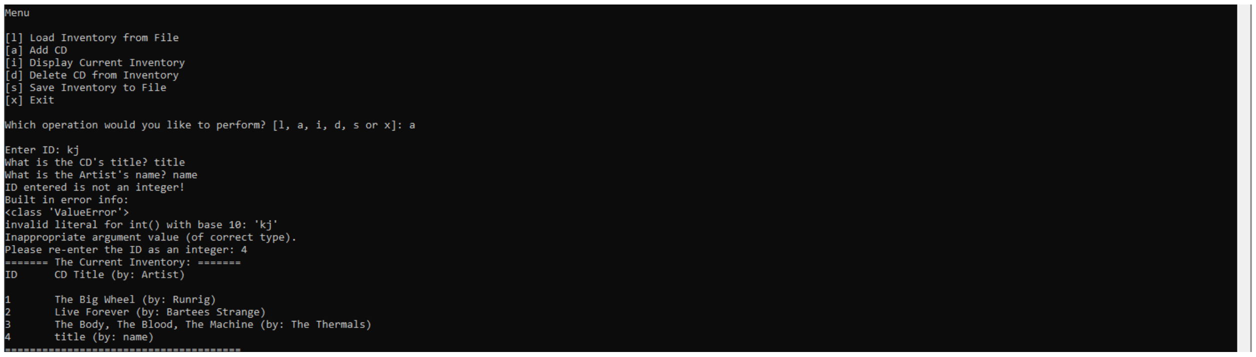


```
€◆•¥        ]”(}”(Œ☻ID”K☻Œ✚Title”Œ
The·Big·Wheel”Œ⯅Artist”Œ⯅Runrig”u}”(h☻K☻h♥ŒLive·Forever”h✚Œ¤Bartees·Strange”u}”(h☻K♥h♥Œ·The·Body,·The·Blood,·The·Machine”h✚ŒThe·Thermals”ue.
```

Figure 8: Spyder Output

Running the Script - Terminal

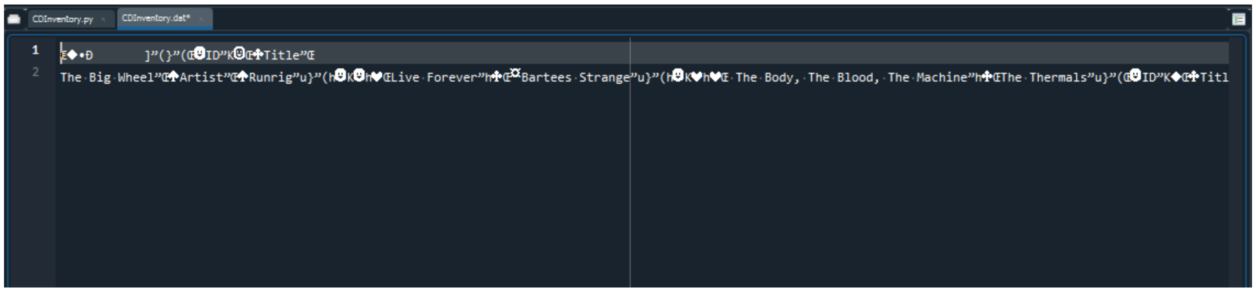
We also ran the program through the Terminal to finish testing. Below you can see the error handling for the input that cannot be converted to a integer.



```
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: kj
What is the CD's title? title
What is the Artist's name? name
ID entered is not an integer!
Built in error info:
<class 'ValueError'>
invalid literal for int() with base 10: 'kj'
Inappropriate argument value (of correct type).
Please re-enter the ID as an integer: 4
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       The Big Wheel (by: Runrig)
2       Live Forever (by: Bartees Strange)
3       The Body, The Blood, The Machine (by: The Thermals)
4       title (by: name)
======================================
```

Figure 9: Terminal ValueError

Below we have the output file after this terminal testing.



```
€◆•Ð       ]”(}”(Œ☻ID”K☻Œ♣Title”Œ
The·Big·Wheel”Œ♠Artist”Œ♠Runrig”u}”(h☻K☻h♥ŒLive·Forever”h♣Œ☼Bartees·Strange”u}”(h☻K♥h♥Œ·The·Body,·The·Blood,·The·Machine”h♣ŒThe·Thermals”u}”(Œ☻ID”K◆Œ♣Titl
```

Figure 10: Terminal Output File

Summary

Given the information provided throughout Module 07, I was able to modify the previous script that helps a user manage a CD inventory by adding error handling and using binary data.