Cheryl Montejo

August 21, 2022

Foundations of Programming, Python

Assignment 7

CD Inventory: Errors and Pickling

# Introduction

For this assignment, I will explain the steps I took to update an existing Python script for a CD inventory system to accommodate error handling as well as use the pickle module for data serialization and deserialization.

# Learning Through the Module

For this assignment, I read the module labs prior to trying the assignment. For this assignment, I used the module to guide me, but I mostly relied on the internet and trial and error to complete the assignment.

# Drafting the Script

For this assignment, we were asked to modify our script from Assignment 6. Based on what was discussed in class on August 17, I updated the functions to pass arguments rather than variables.

For researching error handling, I found W3Schools and TutorialsTeacher to be helpful because they were clear and concise while also including simple examples. (W3Schools, <https://www.w3schools.com/python/python_try_except.asp>) (external site)[[1]](#footnote-1) and (TutorialsTeacher, <https://www.tutorialsteacher.com/python/exception-handling-in-python>) (external site)[[2]](#footnote-2).

I would say the same for RealPython and PythonProgramming.net when I was researching the pickle module (RealPython, <https://realpython.com/python-pickle-module/>)(external site)[[3]](#footnote-3) and (PythonProgramming.net, <https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>) (external site)[[4]](#footnote-4).

For both, I found python documentation to be the least helpful. While the documentation is detailed and thorough, I found it to be too detailed for the purposes of this assignment (Python.org, <https://docs.python.org/3/tutorial/errors.html>; <https://docs.python.org/3/library/pickle.html>)(external site)[[5]](#footnote-5).

When I was updating the script for error handling, I tried to look for situations where the script would be loading in data, converting data, or asking the user for an input. Some of these situations were already covered by existing script, like the loading of data or menu inputs, so I added additional printed text to make it clear to the user what was happening. I did add in some script around the CD ID to force the user to enter in an integer.

Updating the script to use the pickle module was surprisingly easy. I appreciated the tip from class to not overcomplicate loading and dumping the data.

# Running the Script in Spyder and Anaconda Prompt

For running the script in Spyder, I launched Spyder and opened CDInventory.py (File -> Open…->CDInventory.py). The error handling around having an existing .dat file automatically ran when I ran the script. Through the menu, I tested out the options. I started by loading multiple CDs into the inventory and then saving it to a .dat file. I then deleted all the CDs from the script, and then reloaded the inventory from the .dat file before exiting. I ran the script again and loaded the inventory from the .dat file to make sure the inventory would load at the start. I also tested out the error handling for the menu and CD ID inputs. I then deleted the text file so I could start from scratch while testing the script in the Terminal console.

Text

Description automatically generated

Figure 1-Screenshot showing CDInventory.py in Spyder being used to add a CD

Text

Description automatically generated

Figure 2-Screenshot showing CDInventory.py in Spyder being used to add a CD and save the inventory to a .dat file

Graphical user interface, text, application

Description automatically generated

Figure 3-Screenshot showing the data in the .dat file

Text

Description automatically generated

Figure 4-Screenshot showing CDInventory.py in Spyder being used to delete a CD

Text

Description automatically generated

Figure 5-Screenshot showing CDInventory.py in Spyder being used to delete a CD and reload the inventory

Text

Description automatically generated

Figure 6-Screenshot showing CDInventory.py in Spyder being used automatically load the inventory

Text

Description automatically generated

Figure 7-Screenshot showing CDInventory.py in Spyder demonstrating error handling

Text

Description automatically generated

Figure 8-Screenshot showing CDInventory.py in Spyder demonstrating error handling

For running the script in Anaconda Prompt, I opened the Terminal console and navigated to the correct folder using the cd command. I then used the python command with the file name, CDInventory.py, to run the script. Using the menu, I repeated the steps from when I tested the script in Spyder.

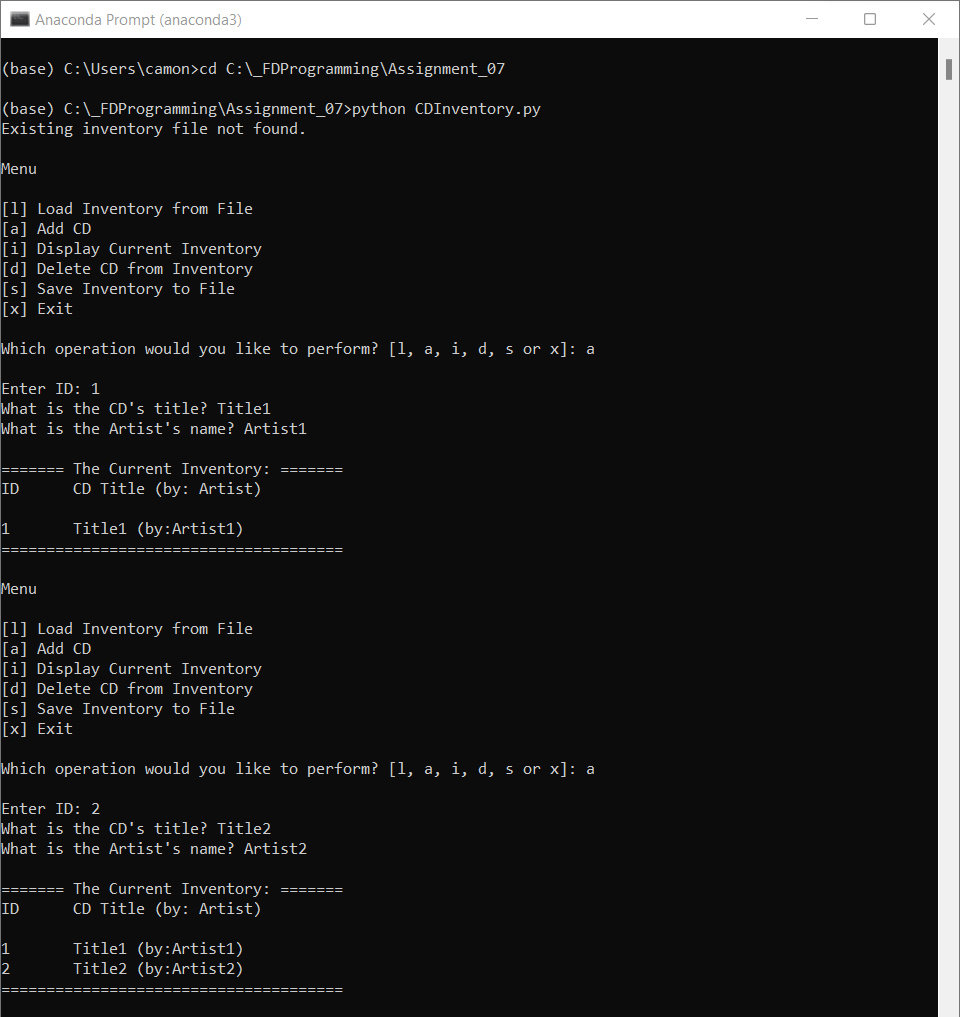


Figure 9-Screenshot showing CDInventory.py in the terminal being used to add a CD

Text

Description automatically generated

Figure 10-Screenshot showing CDInventory.py in the terminal being used save the inventory and then delete a CD

Text

Description automatically generated

Figure 11-Screenshot showing CDInventory.py in the terminal being used to delete a CD and then reload the inventory

Text

Description automatically generated

Figure 12-Screenshot showing CDInventory.py in the terminal to automatically load the inventory

Text

Description automatically generated

Figure 13-Screenshot showing CDInventory.py in the terminal demonstrating error handling

# Summary

In this assignment, we added error handling and used the pickle module for the CD inventory we’ve been working on for the past few assignments. It’s been interesting and fun to continue to build off the same script each week and further refine it to make it a more useable product.

# Appendix

## Link to GitHub Repo

https://github.com/cmonte2uw/Assignment\_07[[6]](#footnote-6)

## Listing AddressBook.py (pasted from Notepad++)

#------------------------------------------#

# Title: CDInventory.py

# Desc: Working with classes and functions.

# Change Log: (Who, When, What)

# DBiesinger, 2030-Jan-01, Created File

# CMontejo, 2022-Aug-13, Copied File

# CMontejo, 2022-Aug-14, Cleaned up formatting and comments, moved TODOs to functions, added error handling to read\_file function

# CMontejo, 2022-Aug-21, Updated functions to use arguments, added error handling around inputs, updated DataProcessor fucnctioms to use pickling

#------------------------------------------#

# -- 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:**

"""Processing the data within the script"""

*@staticmethod*

**def** add\_cd**(**table**,** ID**,** Title**,** Artist**):**

"""Adds new CD to the table

Args:

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

ID (string): ID number for the CD

Title (string): Title of the CD

Artist (string): Artist for the CD

Returns:

None.

"""

dicRow **=** **{**'ID'**:** ID**,** 'Title'**:** Title**,** 'Artist'**:** Artist**}**

table**.**append**(**dicRow**)**

IO**.**show\_inventory**(**table**)**

*@staticmethod*

**def** del\_cd**(**table**):**

"""Deletes a CD from the table

Args:

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

Returns:

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

"""

intRowNr **=** **-**1

blnCDRemoved **=** **False**

**for** row **in** lstTbl**:**

intRowNr **+=** 1

**if** row**[**'ID'**]** **==** intIDDel**:**

**del** lstTbl**[**intRowNr**]**

blnCDRemoved **=** **True**

**if** blnCDRemoved**:**

**print(**'The CD was removed\n'**)**

**return** table

**else:**

**print(**'Could not find this CD!\n'**)**

**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:

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

"""

**try:** # updated error handling to state file not found

table**.**clear**()** # this clears existing data and allows to load data from file

**with** **open(**file\_name**,** 'rb'**)** **as** fileObj**:**

table **=** pickle**.**load**(**fileObj**)**

**print(**'Inventory successfully loaded.\n'**)**

**return** table

**except** **FileNotFoundError:**

**print(**'Existing inventory file not found.\n'**)**

**pass**

*@staticmethod*

**def** write\_file**(**file\_name**,** table**):**

"""Function to manage data ingestion from a list of dictionaries to a file

Reads the data from a 2D table (list of dictionaries) identified as lstTbl into a file.

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.

"""

**with** **open(**file\_name**,** 'wb'**)** **as** fileObj**:**

pickle**.**dump**(**table**,** fileObj**)**

# -- 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:

None.

Returns:

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

"""

choice **=** ' '

**while** **True:**

choice **=** **input(**'Which operation would you like to perform? [l, a, i, d, s or x]: '**).**lower**().**strip**()** #added text to choose valid option

**if** choice **not** **in** **[**'l'**,** 'a'**,** 'i'**,** 'd'**,** 's'**,** 'x'**]:**

**print(**'Please select a valid option.\n'**)**

**else:**

**break**

**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()** # Add extra space for layout

**print(**'======= The Current Inventory: ======='**)**

**print(**'ID\tCD Title (by: Artist)\n'**)**

**for** row **in** table**:**

**print(**'{}\t{} (by:{})'**.format(\***row**.**values**()))**

**print(**'======================================'**)**

**print()** # Add extra space for layout

*@staticmethod*

**def** new\_cd**():**

"""Get user input for a new CD

Args:

None.

Returns:

None.

"""

**while** **True:** #Added error handling to force user to enter an integer for the ID

strID **=** **input(**'Enter ID: '**).**strip**()**

**try:**

intID **=** **(int(**strID**))**

**break**

**except** **ValueError:**

**print(**'Please enter an integer\n'**)**

strTitle **=** **input(**'What is the CD\'s title? '**).**strip**()**

strArtist **=** **input(**'What is the Artist\'s name? '**).**strip**()**

DataProcessor**.**add\_cd**(**lstTbl**,** intID**,** strTitle**,** strArtist**)**

# -- INTERFACE -- #

# 1. When program starts, read in the currently saved Inventory

lstTbl **=** 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 to 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\n'**)**

**if** strYesNo**.**lower**()** **==** 'yes'**:**

**print(**'reloading...'**)**

lstTbl **=** 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 to add a CD

**elif** strChoice **==** 'a'**:**

# 3.3.1 Ask user for new ID, CD Title and Artist and add item to the table (Was a TODO)

IO**.**new\_cd**()**

**continue** # Start loop back at top

# 3.4 Process to display current inventory

**elif** strChoice **==** 'i'**:**

IO**.**show\_inventory**(**lstTbl**)**

**continue** # Start loop back at top

# 3.5 Process to 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

**while** **True:** #Added error handling to force user to enter an integer for the ID

intIDDel **=** **input(**'Which ID would you like to delete? '**).**strip**()**

**try:**

intIDDel **=** **(int(float(**intIDDel**)))**

**break**

**except** **ValueError:**

**print(**'Please enter an integer\n'**)**

# 3.5.2 Search thru table and delete CD (Was a TODO)

DataProcessor**.**del\_cd**(**lstTbl**)**

IO**.**show\_inventory**(**lstTbl**)**

**continue** # Start loop back at top

# 3.6 Process to 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 (Was a TODO)

FileProcessor**.**write\_file**(**strFileName**,** lstTbl**)**

**pass**

**else:**

**input(**'The inventory was NOT saved to file. Press [ENTER] to return to the menu.'**)**

**continue** # Start loop back at top

# 3.7 Catch-all should not be possible, as user choice gets vetted in I/O, but to be safe:

**else:**

**print(**'General Error'**)**

1. Retrieved 2022-Aug-21 [↑](#footnote-ref-1)
2. Retrieved 2022-Aug-21 [↑](#footnote-ref-2)
3. Retrieved 2022-Aug-21 [↑](#footnote-ref-3)
4. Retrieved 2022-Aug-21 [↑](#footnote-ref-4)
5. Retrieved 2022-Aug-21 [↑](#footnote-ref-5)
6. Retrieved 2022-Aug-21 [↑](#footnote-ref-6)