**Mitchell Post**

**FEB-25-2021**

**IT FDN 110 A**

**MOD 7**

# **Introduction**

This week we learned several new tools within python. Using these new tools, we applied our knowledge updating last weeks code with some additional features and capabilities. One of the big topics this week was Structured error handling. Structured error handling in python is the handling and organization of how you want your program to deal with errors. Up to this point in class any errors we say would end the program abruptly and provide an error code followed by a line location of that error. With error handling you can take change the behavior of your program when dealing with errors. For instance, in this assignment we used “try” and “Except” keys to change the behavior of our code. If we try something and it works then the program continues. If we try and it fails, then it can output a text prompting the user to try again. I also so in my research that you can output the error into a file. I think this could be exceptional useful in larger programs handling data, to not stop the program but to output an error then continue or move on.

Another utility learned this week is pickle. With pickle we can convert our data and output it to a binary file that is not human readable. These can be used in python to serialize and store data sets, it is useful because it takes information exactly as it is saved making this tool robust. In this weeks assignment we take our write and read functions from last weeks assignment and modify them to use pickle.load/dump.

I researched pickling in python and found several sites they are attached as reference below. One article I found from datacamp.com was fantastic and provided lots of detail regarding the pickle process. It took me several times to get the syntax correct on this method but once learned it appears to be like a great utility for larger programs

# **Assignment**

* Using weeks 6’s CDInventory.py assignment to perform the following tasks
  + Add structured error handling around the areas where there is user interaction
  + Modify the permanent data store to use binary data.

**Add structured error handling around the areas where there is user interaction**

First thing I did was get last week’s code updated with the suggestions from grading. Most updates were concerning SOC, I’m starting to learn the importance of this. Then I looked at every where in the code where we could use error handling and created a list. With that I started knocking out each one at a time and testing them to ensure they worked properly. I used the “try’ and ‘Except’ key words to manipulate how my program handled errors. I did run into some issues again with SOC and how and where to place my statements. As I understand it “except” by itself will catch errors that are not fatal and handle them as you would like. I’m not sure if we were supposed to identify the errors with that clause but as it is written it seems to function correctly.

**Modify the permanent data store to use binary data.**

In our previous week’s assignments, we introduced a save and load function to our program. We used “open\_file” and “read\_file” to output our table to txt and load back when needed. We had to parse the data and format it so that it would both write correctly then pull the information out of the table correctly. This week pickling was introduced as a way to output and read files. With pickling the file is output into a bit stream and is turned into a serialized object that python can manipulate. This objective took some time for me to find the correct syntax. I first thought that file would have to be parsed from the text similar to how we loaded the data last week. This is not necessary with pickle, as pickling preserves the data in the same state that it was saved as and is readable as such.

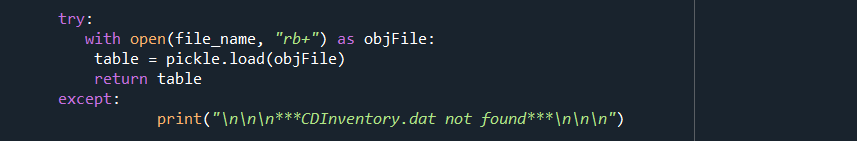


Figure 1 – Pickle Load

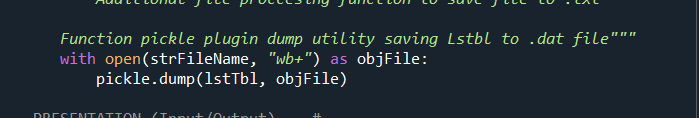


Figure 2 – Pickle dump, Write to file

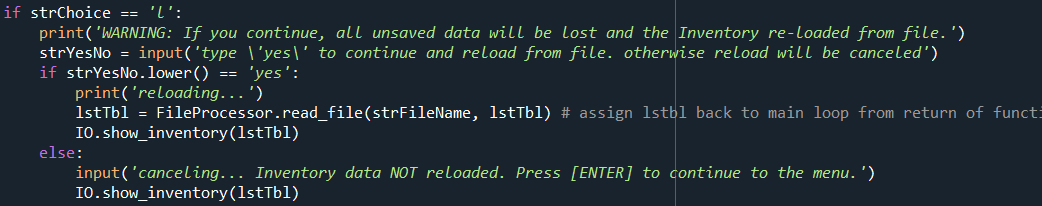
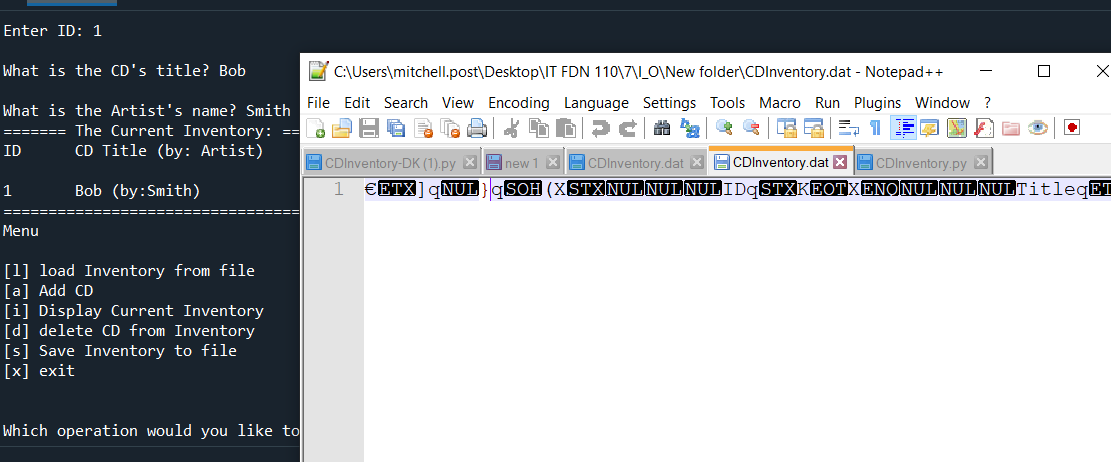


Figure 3 – Main loop pickle load call, passing lstbl back to return value

**Test Functionality of program as whole**

After I had all the concepts down, I was able to complete the assignment and test the program successfully. I found and learned that placement of the I/O handling is important. I also learned about pickling and how it can be used to save and read files easily within code.



*Figure 3 – Testing Functionality .dat output*

**References**

Dawson, M. (2009). *Python® Programming for the Absolute Beginner, Third Edition*. Course Technology PTR.

<https://www.datacamp.com/community/tutorials/pickle-python-tutorial>

https://docs.python.org/3/library/pickle.html

# **Appendix**

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

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

# Title: Assignment07.py

# Desc: Pickles, I/O error Handling.

# Change Log: (Who, When, What)

# POST, 0800-FEB-15, Created File

# POST, 0900-FEB-15, pulled out code for functions and organized for task

# POST, 1100-FEB-15, assigned all code to defintions according to appropriate class

# POST, 1200-FEB-15, troubleshot nested code append

# POST, 1300-FEB-15, Tested code succesfully

# POST, 0815-FEB17, RE ORG FUNCTIONS/CLASS

# POST, 0900-FEB-24, Updated code accordigng to recomandations from grading

# POST, 1200-FEB-24, Added i/o handling

# POST, 1400-FEB-24, Troubleshoot pickle functions - tested succesfully

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

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

# -- DATA -- #

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

value1 = None # is this neccessary? for "def delete\_cd(value1):" Do i have to none out variables

import pickle

# -- PROCESSING -- #

class DataProcessor:

"""Processing the data from input to Dict"""

@staticmethod

def input\_append(lstTbl, strID, strTitle, stArtist): ##Added lstbl pass in per sugestions

"""This function appends the information take from DataProcessor. input\_user(strFileName, lstTbl)

then uses it to append dicrRow

Args:

dicRow: dictionary row.

Table (List of dicts)

Returns:

None

"""

intID = int(strID)

dicRow = {'ID': intID, 'Title': strTitle, 'Artist': stArtist}

lstTbl.append(dicRow)

return lstTbl

@staticmethod

def delete\_cd(lstTbl, id\_to\_remove): ## updated 'value one to id\_to\_remove'

""" Deletion function added here to I/o class

Args:

None.

Returns:

id\_to\_remove User input to select ID to be deleted

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

User input is required to delete desired CD"""

intRowNr = -1

blnCDRemoved = False

for row in lstTbl:

intRowNr += 1

if row['ID'] == id\_to\_remove:

del lstTbl[intRowNr]

blnCDRemoved = True

break

if blnCDRemoved:

print('The CD was removed')

else:

print('Could not find this CD!')

return lstTbl ## Returning list as precuation since updating list

# -- File Proccessing -- #

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 taken from pickle.load this outputs the contents exactly as it was saved

From program.

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.

"""

try:

with open(file\_name, "rb+") as objFile:

table = pickle.load(objFile)

return table

except:

print("\n\n\n\*\*\*CDInventory.dat not found\*\*\*\n\n\n")

@staticmethod

def write\_file(strFileName, lstTbl): # fixed typo

""" Additional file proccesing function to save file to .txt

Function pickle plugin dump utility saving Lstbl to .dat file"""

with open(strFileName, "wb+") as objFile:

pickle.dump(lstTbl, objFile)

# -- 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 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()

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 input\_user():

"""User input taken then assigned to strID, strTitle, and stArtist

Nested function Io.input\_append used to take local variable then

Appended to DICT list

Args:

IO.input\_append(strID, strTitle, stArtist): Function used to take read and append local returns

Returns:

strID = input ('Enter ID: ')

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

stArtist = input ('What is the Artist\'s name? ')

"""

while True:

try:

intID = int(input('Enter ID: ').strip())

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

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

break

except ValueError:

print("Invalid entry. ID must be integer")

return intID, strTitle, stArtist

#### When program starts, read in the currently saved Inventory (no change)

lstTbl = FileProcessor.read\_file(strFileName, lstTbl)

##### start main loop ### (no change)

while True:

IO.print\_menu()

strChoice = IO.menu\_choice()

if strChoice == 'x':

break

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')

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

print('reloading...')

lstTbl = FileProcessor.read\_file(strFileName, lstTbl) # assign lstbl back to main loop from return of function

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 (No Change)

elif strChoice == 'a':

strID, strTitle, stArtist = IO.input\_user()

lstTbl = DataProcessor.input\_append(lstTbl, strID, strTitle, stArtist) ## passed in lstbl this is update from recomendations

IO.show\_inventory(lstTbl)

continue # start loop back at top.

elif strChoice == 'i':

IO.show\_inventory(lstTbl) ## Calls for show function dispalying user inventory

continue # start loop back at top.

elif strChoice == 'd':

try:

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

except:

print("\n\n\n\n\*\*\*\*Invalid CD ID.. Please continue\*\*\*\n\n\n\n")

IO.show\_inventory(lstTbl) ## Calls for show function dispalying user inventory

lstTbl = DataProcessor.delete\_cd(lstTbl, intIDDel) ## Calls function to delete desired CD

continue # start loop back at top.

elif strChoice == 's':

IO.show\_inventory(lstTbl) ## Calls for show function dispalying user inventory

strYesNo = input('Save this inventory to file? [y/n] ').strip().lower()

if strYesNo.lower() in ['yes', 'y']:

FileProcessor.write\_file(strFileName, lstTbl) ## Calls for to save inventory to target .txt file

else:

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

continue # start loop back at top.

### catch-all should not be possible, as user choice gets vetted in IO, but to be save (no change):

else:

print('General Error')