< Weimin Dang >
< Mar-08-2020>
< IT FDN 100 Winter 2020 >
< Assignment07>

Handling errors & binary files

Introduction

Assignment 7 aims at getting me to practice the following knowledge points:

1. Review classes and functions
2. Practice handling errors with try-except
3. Practice working with binary files using the pickle module
4. Continue to practice performing peer review, and using github

The Github repo of this assignment is available at https://github.com/wd-uwGH2020/assignment07

Developing the Code

Modifying Code of Assignment06 using try-except:

A few places in the program are identified where exceptions may occur, such as:

1. In step 1 of the main program: when program starts, read in the currently saved Inventory from file, exceptions may occur when the file can not be found due to incorrect filename & path, file not being created, and file being removed etc. The read_file() function is modified as the following: (the use of pickle in line 120-122 will be discussed in the binary file section of the document)

   It is worth to mention that, although this handles the FileNotFound error, it will result in a None type for the table it would otherwise return if runs succesfully, a None type table is an error itself that will cause future code to crash such as add new inventory or display inventory, therefore, line 129-130 are added to the code.

```
119     try:
120         with open(file_name, 'rb') as objFile: # no file.close() needed using with open
121             table = pickle.load(objFile)
122         return table
123     # try-except to prevent program crash if specified file can't be found due to such as
124     # filename error, file not created, file removed
125     except FileNotFoundError:
126         print("\n\nFile {} was not found and was not able to be loaded\n".format(file_name))
127         # below is necessary otherwise table = None when except is invoked
128         # this function won't crash but the future code will crash
129         table = []
130         return table
```

2. In step 3.3 of the main program: add CD, exceptions may occur because the code requires to convert user input of CD ID from string format to integer format. User could accidentally enter a none number or digital input for ID that will result in a Value error. The add-inventory() function is modified as the following:

```
27      def add_inventory(table):
28          """Add user inputs of new CD to inventory table in memory
36          try:
37              strID = input('Enter ID: ').strip()
38              strTitle = input('What is the CD\'s title? ').strip()
39              strArtist = input('What is the Artist\'s name? ').strip()
40
41              # flexibility of user inputs, e.g. 001 is saved as 1
42              intID = int(strID)
43
44              # make sure user input doesn't result in duplicate ID number
45              intRowNr = -1
46              blnCDExist = False
47              for row in table:
48                  intRowNr += 1
49                  if row['ID'] == intID:
50                      blnCDExist = True
51                      break
52              if blnCDExist:
53                  print("""CD ID '{}' or '{}' already exsits\n
54                        new CD is not added, use a different ID\n""".format(strID, intID))
55              else:
56                  dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
57                  table.append(dicRow)
58                  return table
59
60          # try-except to catch user input ID not in number or digital format
61          except ValueError:
62              print("ID '{}' was not valid and must be digits such as '1' or '01'\n".format(strID))
63
```

I also added the code block between line 44-54 to make sure user input doesn't result in duplicate ID number. This won't cause the system to crash, but will improve the functionality of the program. The mechanics of this section is the same as the delete CD section.

3. In step 3.5 of the main program: delete CD, the same exceptions as add CD may occur because the code requires to convert user input to integer. The mechanics is exactly the same as add CD so I won't repeat here.

Modifying Code of Assignment06 using binary file format:

This applies to the read file and save file section of the code. I modified the read_file() function as the following:

```
106     def read_file(file_name):
107         """Function to manage data ingestion from a binary file to a list
117
118         try:
119             with open(file_name, 'rb') as objFile: # no file.close() needed using with open
120                 table = pickle.load(objFile)
121             return table
```

The write_file() function is modified accordingly as the following:

```
140     def write_file(table, file_name):
141
142         """Function to manage data ingestion from a list to a binary file
151
152         with open(file_name, 'wb') as objFile:
153             pickle.dump(table, objFile)
154
```

Because of these changes, the main program code is also modified where applies, such as:

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

And,

```
244        # 3.2 process load inventory
245        if strChoice == 'l':
246            print('WARNING: If you continue, all unsaved data will be lost and the i
247            strYesNo = input('type \'yes\' to continue and reload from file. otherwi
248            if strYesNo.lower() == 'yes':
249                print('reloading...')
250                lstTbl = FileProcessor.read_file(strFileName)
251                IO.show_inventory(lstTbl)
```

This concludes the tasks of the assignment.

Summary

Running the script in Spyder:

Figure 1 on the next page shows a demonstration of the following operations a user performs:

1. Start the program (a system message returns for non-existence of specified file)
2. Add a new CD (a system message returns for invalid input for ID)
3. Retry add a CD
4. Add a 2nd CD
5. Delete the second entry
6. Save it to the inventory list file
7. Add a 3rd CD
8. Load the saved inventory (the 3rd CD added in 6 is overwritten)
9. Exit the program

Figure 1

```
In [7]: runfile('C:/Users/wdang/Documents/UW Python Course 2020/UW P
Python Course/Jan-Mar Intro/wk7/assignment07')


File CDInventory.dat was not found and was not able to be loaded

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: a

What is the CD's title? Bad

What is the Artist's name? MJ
ID 'a' was not valid and must be digits such as '1' or '01'
```

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


Enter ID: 1

What is the CD's title? Bad

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

1       Bad      (by:MJ)
======================================
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: 02

What is the CD's title? Good

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

1       Bad      (by:MJ)
2       Good     (by:WD)
======================================
```

Figure 1 continues:

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

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

1       Bad      (by:MJ)
2       Good     (by:WD)
=======================================

Which ID would you like to delete? 2
The CD was removed

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

1       Bad      (by:MJ)
=======================================
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)

1       Bad      (by:MJ)
=======================================

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

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


Enter ID: 3

What is the CD's title? WW

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

1       Bad      (by:MJ)
3       WW       (by:EE)
=======================================
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 canceledyes
reloading...
======= The Current Inventory: =======
ID      CD Title (by: Artist)

1       Bad      (by:MJ)
=======================================
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]: x


In [8]:
```

Figure 2 shows the same code and process running in the terminal.

Figure 2

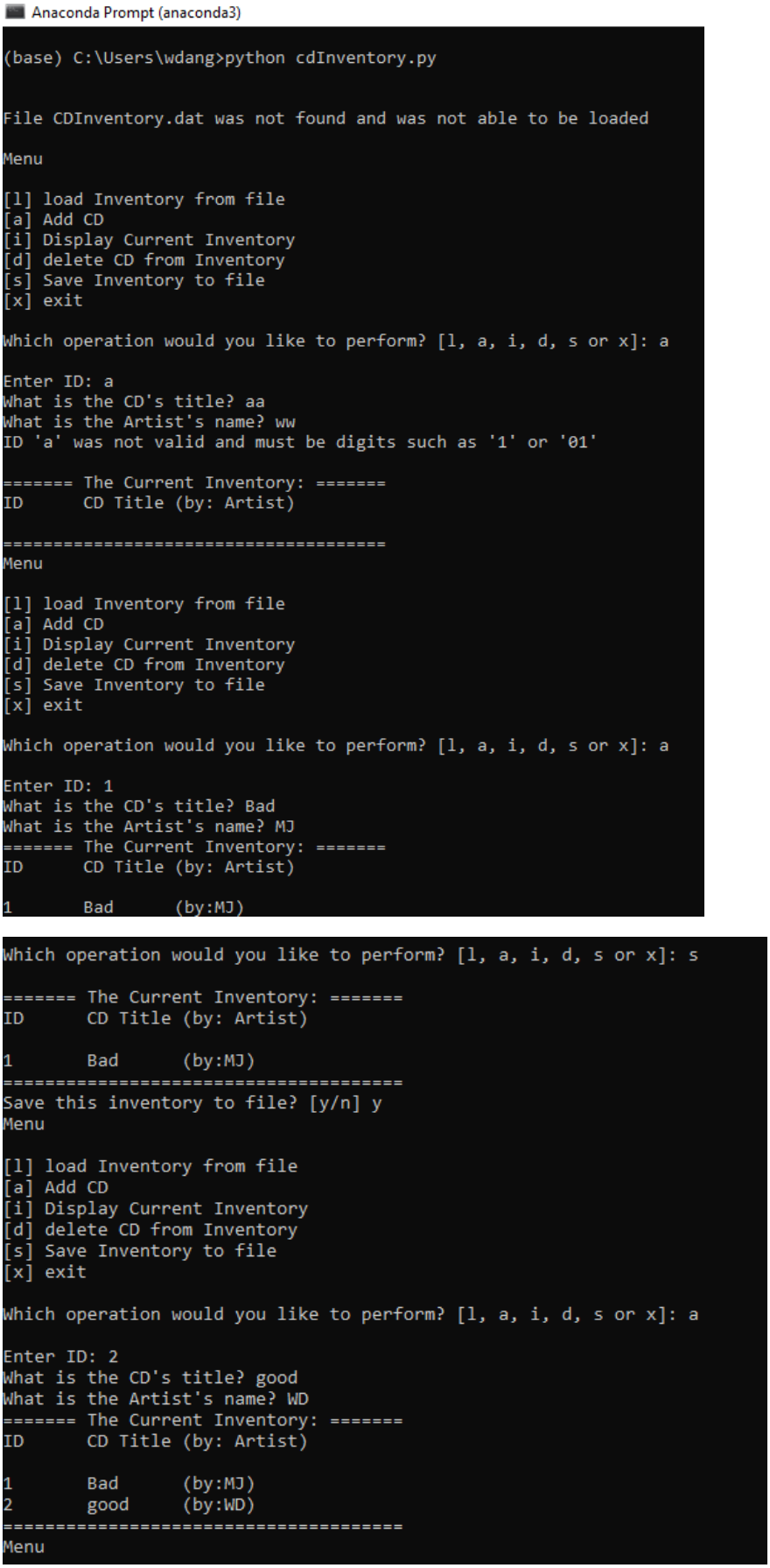
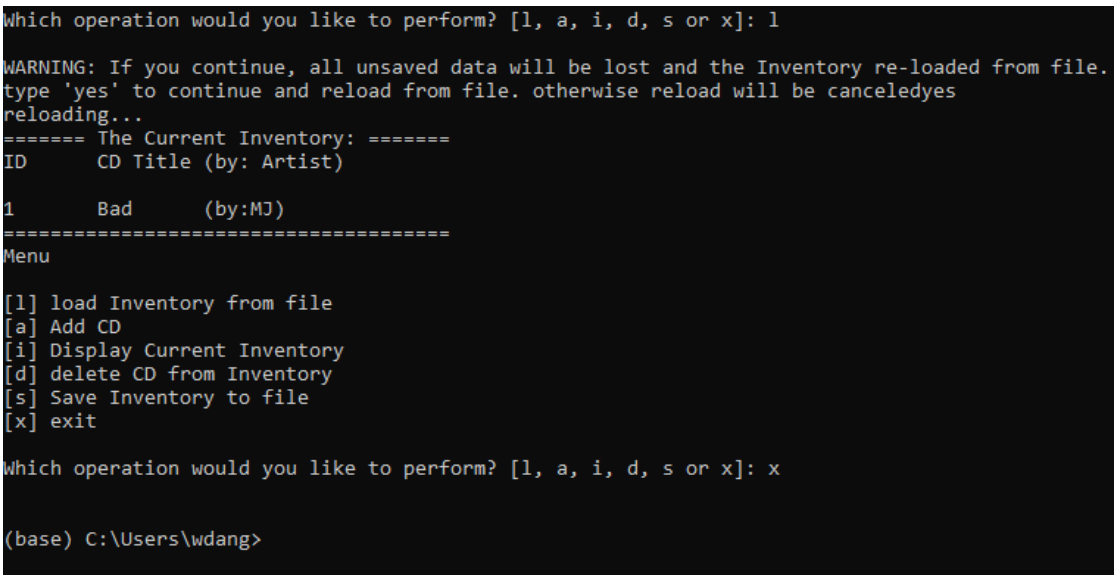


Figure 2 continues:



Lessons learned:

Handling errors with exception:

A key lesson I learned through this assignment is although using try-exception can prevent system from crashing due to predictable errors that may occur, it is important to realize sometimes it could result in unintended consequences such as returning a None type variable which could cause the rest of the code to crash.

Working with binary format files:

The benefits of using binary format files include reduced file size, processing time and use of memory to convert human readable format. The cautions of using the pickle module include non-human readable is NOT encrypting and unpickling file of unknow source is unsecure.[1]

[1] https://docs.python.org/3/library/pickle.html

Appendix – copy of the script

```
#------------------------------------------#
# Title: CDInventory.py
# Desc: Working with files and handling exceptions.
# Change Log: (Who, When, What)
# DBiesinger, 2030-Jan-01, Created File
# Wdang, 2020-Feb-28, Modified file for assignment06
# Wdang, 2020-Mar-08, Midified file for assignment07
#------------------------------------------#

""" This program is a menu operated CD inventory system """

import pickle # To save and read binary format file

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


# -- PROCESSING -- #
class DataProcessor:
    """Processing data in memory"""

    @staticmethod
    def add_inventory(table):
        """Add user inputs of new CD to inventory table in memory

        Args:
            table: the list of dictionaries containing the CD entries

        Returns:
            the modified list of dictionaries with new entries of CDs
        """
        try:
            strID = input('Enter ID: ').strip()
            strTitle = input('What is the CD\'s title? ').strip()
            strArtist = input('What is the Artist\'s name? ').strip()

            # flexibility of user inputs, e.g. 001 is saved as 1
            intID = int(strID)

            # make sure user input doesn't result in duplicate ID number
            intRowNr = -1
            blnCDExist = False
            for row in table:
                intRowNr += 1
                if row['ID'] == intID:
                    blnCDExist = True
                    break
            if blnCDExist:
                print("""CD ID '{}' or '{}' already exsits\n
                      new CD is not added, use a different ID\n""".format(strID,
intID))
            else:
```

```
                dicRow = {'ID': intID, 'Title': strTitle, 'Artist': strArtist}
                table.append(dicRow)
                return table

        # try-except to catch user input ID not in number or digital format
        except ValueError:
            print("ID '{}' was not valid and must be digits such as '1' or
'01'\n".format(strID))

    @staticmethod
    def delete_inventory(table):
        """Delete CD from inventory table in memory based on
        user input CD ID and confirm the deletion with user


        Args:
            table: the list of dictionaries containing the CD entries.
            ID: the ID of the CD to be deleted

        Returns:
            the modified list of dictionaries with removal of entries matching CD
ID entered

        """
        IDDel = input('Which ID would you like to delete? ').strip()

        try:
            intIDDel = int(IDDel)

            intRowNr = -1
            blnCDRemoved = False
            for row in table:
                intRowNr += 1
                if row['ID'] == intIDDel:
                    del table[intRowNr]
                    blnCDRemoved = True
                    break
            if blnCDRemoved:
                print('The CD was removed\n')
            else:
                print('Could not find this CD!\n')
            return table

        # try-except to catch user input ID not in number or digital format
        except ValueError:
            print("ID '{}' was not valid and must be digits such as '1' or
'01'\n".format(IDDel))


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

    @staticmethod
    def read_file(file_name):
        """Function to manage data ingestion from a binary file to a list

        Reads the data from binary file identified by file_name into a list.
```

```
        Args:
            file_name (string): name of file used to read the data from

        Returns:
            a list contains file content.
        """

        try:
            with open(file_name, 'rb') as objFile: # no file.close() needed using
with open
                table = pickle.load(objFile)
            return table
        # try-except to prevent program crash if specified file can't be found
due to such as
        # filename error, file not created, file removed
        except FileNotFoundError:
            print("\n\nFile {} was not found and was not able to be
loaded\n".format(file_name))
            # below is necessary otherwise table = None when except is invoked
            # this function won't crash but the future code will crash
            table = []
            return table


# alternative code for not reading in binary format, maybe useful for future
        # for line in data:
        #     id = line.strip().split(',')
        #     dicRow = {'ID': int(id[0]), 'Title': id[1], 'Artist': id[2]}
        #     table.append(dicRow)


    @staticmethod
    def write_file(table, file_name):

        """Function to manage data ingestion from a list to a binary file

        Args:
            table: data in memory
            file_name: name of file used to save the data to

        Returns:
            None.
        """

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


# alternative code for not saving in binary format
    # Overwrite the data from file identified by file_name with the follow:
    # Extract the values only of the each row of the list of dicts as a new list,
one row a time
    # Convert the first value to string format
    # Write each of the values comma separated, and start a new line at the end
```

```
        # objFile = open(file_name, 'w')
        # for row in table:
        #     lstValues = list(row.values())
        #     lstValues[0] = str(lstValues[0])
        #     objFile.write(','.join(lstValues) + '\n')
        # objFile.close()


# -- 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 = input('Which operation would you like to perform? [l, a, i, d, s
or x]: ').lower().strip()
        while choice not in ['l', 'a', 'i', 'd', 's', 'x']:
            print("'{} is not a valid operation".format(choice))
            choice = input('Which operation would you like to perform? [l, a, i,
d, s or x]: ').lower().strip()
        print()  # Add extra line for layout
        return choice

    @staticmethod
    def show_inventory(table):
        """Displays data in memory, only shows the values of the dictionaries


        Args:
            table: data in memory.
```

```
        Returns:
            None.

        """
        print('======= The Current Inventory: =======')
        print('ID\tCD Title (by: Artist)\n')

        for row in table:
            print('{}\t{}\t (by:{})'.format(*row.values()))
        print('=======================================')



# start main program

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

# 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')
        if strYesNo.lower() == 'yes':
            print('reloading...')
            lstTbl = FileProcessor.read_file(strFileName)
            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 strChoice == 'a':
        # 3.3.1 Ask user for new ID, CD Title and Artist and add to CD inventory
table
        DataProcessor.add_inventory(lstTbl)
        # 3.3.2 Display the updated inventory list
        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 display Inventory to user
        IO.show_inventory(lstTbl)
        # 3.5.2.1 get user input for which CD to delete
        # 3.5.2 2 search thru table and delete CD if found
        DataProcessor.delete_inventory(lstTbl)
        # 3.5.3 display updated Inventory to user
        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(lstTbl, strFileName)
        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 IO, but
to be safe:
    else:
        print('General Error')
```