Kenneth Chiu

March 13, 2022

Foundations of Programming: Python

Assignment 07

The Magic CD Inventory Program - Continued

# Introduction

In this module, we learned about working with text files using different modes, reading data using various methods, working with binary data using the Pickle module and error/exception handling. We implemented several things we learned in this module to add more “flavors” to the Magic CD Inventory Program.

# What is Exception/Error Handling?

Exception/error handling is a way to prevent a program from crashing abruptly when errors arise and still allows the program to run until the end. We use the try/except blocks to handle various types of errors anticipated in the program. The common errors we’ve encountered so far are FileNotFoundError, ValueError and ZeroDivisionError. After searching on the Internet, I found several websites providing a lot of materials on exception/error handling and the followings are the ones I found particularly helpful:

* Python.org - The Python Tutorial
  + <https://docs.python.org/3/tutorial/errors.html>
  + <https://docs.python.org/3/library/exceptions.html#bltin-exceptions>
  + This website provides a very comprehensive documentation on exception/error handling and some useful tips when implementing. For example, it’s an useful convention to provide the exception type in the error message, such as the FileNotFoundError, when creating user-defined exceptions. Additionally, this website also dives deeper into the clean-up actions such as “finally” clause and “with” statements which is briefly mentioned in Module 7 material. However, with so much details, it’s a bit overwhelming when reading through the entire documentation but it’ll become more and more useful as I become a more season Python coder.
* Programiz
  + <https://www.programiz.com/python-programming/exception-handling>
  + This website does a great job explaining the exception/error handling concept by providing a video instruction and with simple examples which are very useful and friendly for beginners.
* w3schools
  + <https://www.w3schools.com/python/gloss_python_error_handling.asp>
  + <https://www.w3schools.com/python/python_try_except.asp>
  + I really enjoyed the interactive interface on the w3schools site which is very helpful to visualize the examples with the built-in practice IDE. This site also explains with simple examples which are easily digestible. One downside of this website is that it appears to consume a lot of memory which seems to significantly slow down my computer performance.

# What is Pickling in Python?

Pickling is the process of serializing objects so they can be easily stored in the binary format. And to read in data in binary format , we can also unpickle the objects and deserialize the objects back to be readable contents. Below are the three websites that I found on the internet discussing the concept of pickling/serialization and unpickling/deserialization.

* Realpython
  + <https://realpython.com/python-pickle-module/>
  + This article starts by discussing the concept of serialization and various types of Python modules, including the Pickle module, that support this functionality. This site dose a good job laying the groundwork and explaining the basic concept and functions and also provides a simple example which is easily digestable by beginners. The discussion ultimately gets into more complex functions and concepts which are beyond my comprehension at this point.
* Python.org
  + <https://docs.python.org/3/library/pickle.html>
  + Similar to the Realpython article, the Python.org documentation also discusses the basic concept of serialization, different modules and functions; however, it soon covers the concepts that are way beyond my comprehension at this point.
* Towardsdatascience
  + <https://towardsdatascience.com/do-not-use-python-pickle-unless-you-know-all-these-facts-d9e8695b7d43>
  + In addition to the pictures of delicious pickled vegetables, this article provides very simple and step-by-step examples which makes it very easy to follow. It also discusses one of the main advantages of pickling data, which is speed, and provides a comparison with read/write functions with other data types.

# The Magic CD Inventory program

GitHub link - https://github.com/kennchiu/Assignment\_07

To add more “flavors” to the Magic CD Inventory program, I loaded the CDInventory.py file from Assignment 6 and updated the script header – List 1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | *#------------------------------------------#*  *# Title: CDInventory.py*  *# Desc: Working binary data and error/exception handling.*  *# Change Log: (Who, When, What)*  *# DBiesinger, 2030-Jan-01, Created File*  *# KChiu, 2022-Mar-06, Updated codes to complete the Magic CD Inventory Program*  *# KChiu, 2022-Mar-13, Updated program to save output and read input as binary data and to add error handling functions.*  *#------------------------------------------#* |

List - Script Header

Next, to manage data in binary format, I imported the Pickle module and update the strFileName file type to be .dat in the DATA section of the program – List 2.

|  |  |
| --- | --- |
| 10  11  12  13  14  15  16 | *# -- DATA -- #*  **import** **pickle** *# import modoule needed to pickling/unpickling*  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* |

List - Import Pickle Module

In the FileProcessor Class, for the read\_file() function, I updated the code on rows 86 and 87 to read in CDInventory file using pickle.load() function. Additionally, I also converted this part to a try/except block in order to alert the user when the CDInventory.dat file does not exist. I decided to display a customized error message as well as the Python built-in error message as you see on row 92 and 93 – List 3.

For the write\_file() function, I updated the codes on row 110 and 111 to use the pickle\_dump() function to write the CD Inventory table to the CDInventory file in binary format. For both functions, I updated the docstrings to indicate the functionality of managing binary data – List 3.

|  |  |
| --- | --- |
| 66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112 | **class** **FileProcessor**:  *"""Processing the data to and from text file"""*  @staticmethod  **def** read\_file(file\_name, table):  *"""Function to manage binary data ingestion from file to a list of dictionaries*  *Reads the data in binary format 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 binary data from*  *table (list of dict): 2D data structure (list of dicts) that holds the data during runtime*  *Returns:*  *None.*  *"""*  table.clear() *# this clears existing data and allows to load data from file*  *# read in CD inventory data in binary format with pickle module*  **try**:  **with** open(file\_name, 'rb') **as** objFile:  dill\_pickle = pickle.load(objFile)  **for** row **in** dill\_pickle:  dicRow = {'ID': int(row['ID']), 'CD Title': row['CD Title'], 'Artist': row['Artist']}  table.append(dicRow)  **except** **FileNotFoundError** **as** e:*# added error handling if CDInventory file doesn't exist.*  print('**\n**FileNotFoundError: ' + file\_name +' File does not exist!')  print(e, '**\n**')  @staticmethod  **def** write\_file(file\_name, table):  *"""Function to sync the data in memory to file by saving current table in binary format*    *Writes the data in the lstTbl in memory to the CDInventory.txt file in binary format.*    *Args:*  *file\_name (string): name of file used to write/save the data to*  *table (multiple lists): 2D data structure (values of multiple lists of dicts)*  *that holds the current data during runtime*  *Returns:*  *None.*    *"""*  *# save data in binary format with pickle module*  **with** open(file\_name, 'wb') **as** objFile:  pickle.dump(table, objFile) |

List - Class FileProcessor

In the PROCESSING section, I added a try/except block in the Add a CD step to handle the error if user provides an input that cannot be converted to an integer and display both a customized error and Python built-in message for ValueError type of error – List 4.

|  |  |
| --- | --- |
| 191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208 | *# 3.3 process add a CD*  **elif** strChoice == 'a':  *# 3.3.1 Ask user for new ID, CD Title and Artist*  **while** **True**:  *#Convert ID to integer format since default user input is in string format*  strID = input('Enter ID: ').strip()  **try**:  intID = int(strID)  **break**  **except** **ValueError** **as** e: *# added error handling if user provides a value that can't be converted to an integer*  print('ValueError: This is not an integer!')  print(e, '**\n**')  **continue**  strTitle = input('What is the CD**\'**s title? ').strip()  strArtist = input('What is the Artist**\'**s name? **\n**').strip()  DataProcessor.add\_cd(strID, strTitle, strArtist)  IO.show\_inventory(lstTbl)  **continue** *# start loop back at top.* |

List - Add a CD

In the Delete a CD step, I added another try/except block to handle the error if user provides a CD ID that cannot be converted to an integer and display both a customized error and Python built-in message for ValueError type of error – List 5.

|  |  |
| --- | --- |
| 213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229 | *# 3.5 process delete a CD*  **elif** strChoice == 'd':  *# 3.5.1 get User input 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**:*# continues to ask until an integer is provided.*  **try**:  intIDDel = int(input('Which ID would you like to delete? ').strip())  **break**  **except** **ValueError** **as** e:*# added error handling if user provides a value that can't be converted to an integer*  print('**\n**ValueError: This is not a valid CD ID! Please enter an integer.')  print(e, '**\n**')  **continue***# start loop back and ask for an ID number.*  DataProcessor.delete\_cd(intIDDel)  IO.show\_inventory(lstTbl)  **continue** *# start loop back at top.* |

List - Delet a CD

# Summary

Learning how to work with data in binary format is both interesting and confusing. It took me many tries to realize that you can’t convert a file to be in binary format by simply changing the file extension to “.dat”. Instead, I need to create a binary file by writing something with the dump() function first then the file can be read back in. I’m still not certain what a use case would be for saving data in the binary format but I’m sure this won’t be the last time I hear about this.

Everything below is required screenshots per assignment instructions.

Below are screenshots showing the CD Inventory program working on Spyder – Figure 1 through 9.

Text

Description automatically generated

Figure - Menu with FileNotFoundError Message

Text

Description automatically generated

Figure - Add CD 1

Text

Description automatically generated

Figure - Add CD 2

Text

Description automatically generated

Figure - Add CD 3

Text

Description automatically generated

Figure - Display Current Inventory

Text

Description automatically generated

Figure - Delete CD 3

Text

Description automatically generated

Figure - Save CD Inventory to CDInventory.dat File

Text

Description automatically generated

Figure - Exit Program

Graphical user interface, text, application

Description automatically generated

Figure - CDInventory File after Running through Spyder

Below screenshots show Magic CD Inventory program run in the Terminal – Figure 10 through 14.

Text

Description automatically generated

Figure - CD Inventory Program via Terminal - Part 1

Text

Description automatically generated

Figure - CD Inventory Program via Terminal - Part 2

Text

Description automatically generated

Figure - CD Inventory Program via Terminal - Part 3

Text

Description automatically generated

Figure - CD Inventory Program via Terminal - Part 4

Graphical user interface, text, application, email

Description automatically generated

Figure - CDInventory File after Running through Terminal