Jerry Hsieh

Mar-12-2022

Foundations of Programming: Python

Assignment 07

Structured Error Handling and Binary Files

# Introduction

This is a script which we need to modify our Assignment 06 script as required to add Structured Error Handling and save the data as binary files. I studied the Module\_07 materials and tried to understand these concepts and make the script work.

# Request of the assignment

There are several requests for this assignment. What we need in the assignment are:

1. Search the web for examples of how to use error handling feature.
2. Search the web for examples of how to use Pickling feature.
3. Structured error handling for:
   1. User interaction
   2. Type case (string to int)
   3. File access.
4. Data storage using binary data.

# Error Handling example

I found an interesting website called “Geek University” which explains the error handling “try...except…else…” clauses very well.

[The try...except...else statements | Python# (geek-university.com)](https://geek-university.com/the-try-except-else-statements/)

There is an example here.

Text

Description automatically generated

Figure 1 - Error handling example

If we execute the script, it will run the “try” clause if there is no exception such as entering non-numeric value. If a non-numeric value is detected, it will end up with “except” clause, which print out the error message. If we do enter a numeric value, it will go to “else” clause and run the if-statement. Trying to use “else” in script can provide more flexibility in coding work.

# Binary data storage

In Stack Overflow website, I found a useful example of how to use pickling feature to save a dict data:

[dictionary - How can I use pickle to save a dict (or any other Python object)? - Stack Overflow](https://stackoverflow.com/questions/11218477/how-can-i-use-pickle-to-save-a-dict-or-any-other-python-object)

There is an example here. Basically, the concept is like Module\_07 document. It is easier to dump the dict dataset to a .dat file (binary data) then load it. Instead of reading and writing row by row, we could use pickle.dump to save everything to .dat, then pickle.load everything out from this .dat file. Output will still be a dict dataset.

Text

Description automatically generated

Figure 2 - Pickling example

# Explanation of the script

1. Structured error handling: User Interaction. In menu choice function, if user’s input is not in [l,a,I,d,s,x], we are going to raise Exception to show the user “Not a valid choice”. This while-loop will keep running until user enter any choice which is in [l,a,I,d,s,x]. (Figure 3)

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure 3 - Script for user interaction- main choice

In delete file function, if what user would like to delete is not in data inventory, it will raise Exception to remind the user to enter other value to delete again. (Figure 4)

Text

Description automatically generated

Figure 4 - Script for user interaction – delete

1. Structured error handling: Type case (string to int). If user’s input is not an integer, “ValueError” will occur and print out message to remind user to input again. (Figure 5)

Text

Description automatically generated

Figure 5 – Script for type case error

1. Structured error handling: File access. If there is no file for saving, “FileNotFoundError” will occur and print out the message to remind user adding a file. (Figure 6)

Text

Description automatically generated

Figure 6 - Script for file access error

1. Binary data storage. Use pickle function to save (Figure 7) or load the data. (Figure 8)

Text

Description automatically generated

Figure 7 - pickle.dump()

Text

Description automatically generated

Figure 8 - pickle.load()

# Save your script

File path of my script is C:\\_FDP\Assignment07\CDInventory.py (Figure 9)

Graphical user interface, application

Description automatically generated

Figure 9 - File path

# Run your script/Verify correct functioning

I used Spyder to run this script first. Here I start from testing Structured Error Handling first.

1. I removed CDInventory.dat file to test. It showed “File Not Found” then terminate the program. (Figure 10)

Text

Description automatically generated

Figure 10 - File access test

1. I moved CDInventory.dat file back and tried to test user interaction. I entered “yyyy” intentionally to test and “invalid choice” occurred. We need to make sure our input is in [l,a,I,d,s,x]. (Figure 11)

Text

Description automatically generated

Figure 11 - User interaction test

1. I tried to enter a CD ID as non-numeric value “a”, error message occurred again. After I enter “3”, the script could work without error message. (Figure 12)

Text

Description automatically generated

Figure 12 - Type case test

1. I tried to delete the CD data I just entered but gave it a wrong CD ID number. Error message occurred and wanted me to try it again. (Figure 13)

Text

Description automatically generated

Figure 13 - CD deletion test

1. I entered ‘s’ to save 2 CD datasets as binary data. It did generate a .dat file. The file is not human-readable. (Figure 14)

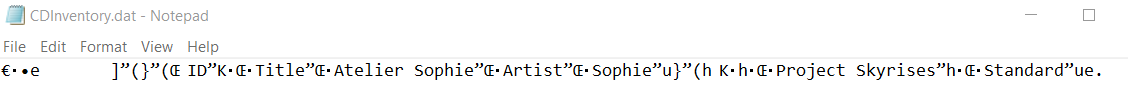


Figure 14 - Binary data storage

1. I entered ‘l’ to load this binary data file. It showed the data correctly. (Figure 15)

Text

Description automatically generated

Figure 15 - Binary data loading

# Run your script (Terminal Window)

I used Anaconda Prompt to run the scripts again with different data input.

1. CDInventory.dat missing test: (Figure 16)

Graphical user interface, text, application

Description automatically generated

Figure 16 -File access test (Terminal)

1. User input error test: (Figure 17)

A screenshot of a computer

Description automatically generated with medium confidence

Figure 17 - User interaction test (Terminal)

1. Input non-numeric value test: (Figure 18)

Text

Description automatically generated

Figure 18 - Type case test (Terminal)

1. Delete wrong CD ID number test: (Figure 19)

Text

Description automatically generated

Figure 19 -CD deletion test (Terminal)

1. Save binary data: (Figure 20)

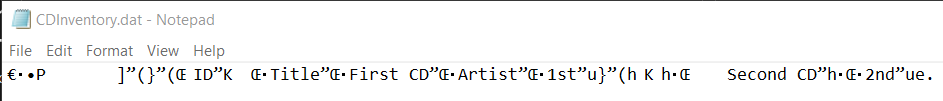


Figure 20 - Binary data storage

1. Load binary data: (Figure 21)

Text

Description automatically generated

Figure 21 - Binary data loading

The scripts work perfectly!

# GitHub link for Assignment

[chhsieh0630/Assignment\_07: IT FDN 110 B Wi 22: Foundations Of Programming: Python (github.com)](https://github.com/chhsieh0630/Assignment_07)

# Summary

In this assignment, I learned how to do Structured Error Handling. This is an important function because humans always make lots of errors. We need fool-proof designs for everything to prevent human from doing things wrong. Try and except clauses are very powerful here since they guided people what to do if they keep doing things incorrectly. Binary data storage is also very useful. We don’t need to read or save the data line by line. Just dump everything as binary form and pull everything out, then work done! This is a very good practice for me in the future to handle data such as this.