

Trevor Huynh

Nov 29, 2022

IT FDN 110 A

Assignment 07

<https://github.com/trevwin/IntroToProg-Python-Mod07>

<https://github.com/trevwin/IntroToProg-Python-Mod07/blob/main/README.md>

Structured Error Handling / Pickling

1.0 Introduction

In this week, the topics covered were structured error handling / pickling.

In this assignment, structured error handling and pickling will be explained along with code demos to show their functionality. A final script will be presented that incorporates both demos into a combined script based on a modification of assignments 05, 06, and Lab 7-1.

2.0 Python Error Types

There are 2 types of errors in Python: syntax and exception errors [2]. Syntax errors occur when the code has incorrect statements (ie. mismatched brackets). Exception errors occur when syntactically correct Python code results in an error [3]. In this assignment, exception errors will be covered.

3.0 Basic Exception Error Catching

Although Python has a standard library of built-in exceptions, it sometimes is not intuitive to the user at first glance. Understanding the standard exception error in Python may require going into the official Python 3 documentation and looking into the specific error class. Structured error handling allows for the user to display more easy to follow error descriptions for diagnosing errors in Python code.

The most basic form of error handling comes in the form of a try-except statement. The try clause executes the statement between the try and except keywords. If there is no exception, the code skips the except block. If there is an exception (ie. exception error), the except block then executes. Without this basic exception handling, the exception error could crash the script. As this is a basic method to catch an exception error, one of the issues is that it hides all errors and catches it with this basic clause. Essentially, you wouldn't be able to tell if you had a Python built-in ZeroDivisionError or a FileNotFoundError as the except block catches all of the errors.

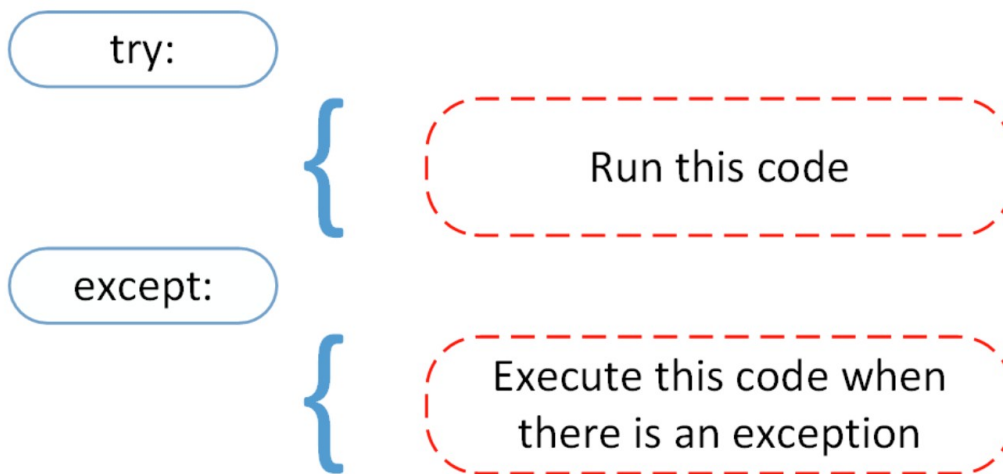


Figure 1. Try Except Block Summary. Taken from [3]

4.0 Custom Exception Error Catching

The basic try-except block can be made more sophisticated in order to catch specific errors.

Python automatically creates an Exception object with error information when an error occurs [4]. This can be captured in a try-except block with error information extracted out.

In addition, catching specific errors can be done by “raising” an exception. Raising an exception creates an exception based on a specific condition [3].

4.1 Demo 1 – Error Catching

In this demo, error handling will be applied to the user inputs section of the script. This demo is similar to the HomeInventory script in assignments 05 and 06 where the user inputs the name of an item and it's associated value.

```
8 # Error Handling Demo
9 strName = str(input("Enter an item name: "))
10 intValue = int(input("Enter a value ($): "))
11
12 lstNameValue = [intValue, strName]
13 print(lstNameValue)
```

Figure 2. User Inputs Script

The anticipated error here is if the user enters a string for the value. At first glance, it is not possible to know what kind of exception object will be created so before doing the error handling, the script is first run by entering a string for the value of the item.

```

Enter an item name: 123
Enter a value ($): hello
Traceback (most recent call last):
  File "C:\_PythonClass\Module07 - Files and Exceptions\test7.py", line 10, in <module>
    intValue = int(input("Enter a value ($): "))
ValueError: invalid literal for int() with base 10: 'hello'

Process finished with exit code 1

```

Figure 3. Exception Error - ValueError

We can see here that the since the item name is a string variable, it can take integer inputs and will not result in an error. Conversely, we can also see that entering a string input to the value of the item then results in an error that crashes the script.

The exception object created by the error of entering a string for the value is exception object ValueError. We can then use this to deal with this specific exception.

The addition of an if-else statement with the isnumeric() function can also catch if the user does not enter a proper item name. Also, I decided to raise a custom exception for this specific case to contain the numeric error.

```

8  # Error Handling Demo
9  try:
10     strName = str(input("Enter an item name: "))
11     if strName.isnumeric():
12         raise Exception("Please do not use numbers!")
13     else:
14         intValue = int(input("Enter a value ($): "))
15         lstNameValue = [intValue, strName]
16         print(lstNameValue)
17     except ValueError as e:
18         print("Please enter a number for the item's value!")
19     except Exception as e:
20         print() # space for looks
21         print("There was a non specific error!")
22         print() # space for looks
23         print("Built in Python error info: ")
24         print(e, sep='\n')

```

Figure 4. Completed Error Handling for User Input

```
Enter an item name: 123

There was a non specific error!

Built in Python error info:
Please do not use numbers!

Process finished with exit code 0
```

Figure 5. Error Handling Result (ValueError)

5.0 Pickling

Pickling is a method to store data in a binary format that obscures the data [4]. This means it converts Python objects into a binary data. The advantage of pickling is to allow for easy data transfer from one system to another and then store it in a file or database [5].

To pickle data, the pickle library must be imported into Python. Two functions `save_data_to_file`, `read_data_from_file` were defined to save data into binary (pickle) and to read the binary file as text (unpickle).

Pickling and unpickling requires pickle library specific functions `pickle.dump()` and `pickle.load()` which convert data into binary and convert binary to text respectively.

5.1 Demo 2 - Pickling

Two functions were created based on Lab 7-1 to demonstrate this.

```

strFileName = 'A07.dat'
lstItemValue = []
# -----PROCESSING-----#
# Pickling Demo
def save_data_to_file(file_name, list_of_data):
    """
    This function saves a list of data to a binary file.

    :param file_name: Name of binary file (.dat format)
    :param list_of_data: List of user data
    :return: No return variable, this function just saves the data to the file
    """

    file = open(file_name, "ab") # ab = a refers to append, b refers to binary
    pickle.dump(list_of_data, file) # pickle.dump() function means to "dump" data into the file. Pickle specific
    file.close()

```

Figure 6. Saving Data as Binary to File Function

```

def read_data_from_file(file_name):
    """
    This function reads existing binary data from the binary file

    :param file_name: Name of binary file
    :return: Returns list_of_data which is converted from binary to text
    """

    file = open(file_name, "rb") # rb = r refers to read, b refers to binary
    list_of_data = pickle.load(file) # pickle.load() function means to read the binary file and unpickle the contents
    file.close()

    return list_of_data

```

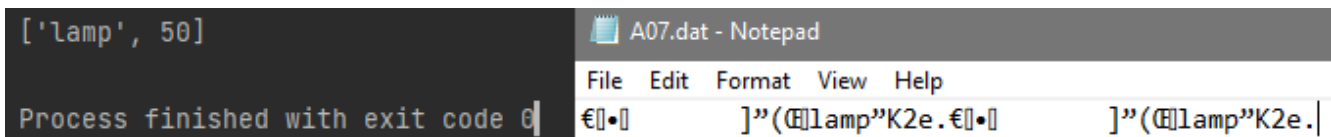
Figure 7. Reading Data From Binary to Text Function (one line)

```

# Main Script
lstItemValue = ["lamp", 50]
save_data_to_file(strFileName, lstItemValue)
print(read_data_from_file(strFileName))

```

Figure 8. Demo 2 – Main Script



The image shows two windows from a PyCharm IDE. The bottom window is the 'Run' console, displaying the output of the script: `['lamp', 50]` and `Process finished with exit code 0`. The top window is a Notepad application titled 'A07.dat - Notepad', showing the binary data saved to the file, which appears as a garbled string: `]”(lamp”K2e.€•`.

Figure 9. PyCharm Output of Script and Binary Data

6.0 Combined Structured Error Handling and Pickling

A script was created using elements of structured error handling and pickling as seen in the above sections. The script asks for the user to input an item name, and then enter a corresponding numerical value for the item's value in \$. The list data is then stored in a binary file where it can then be retrieved. The main body of the script utilizes an outer while loop to guide the user through the menu options as previously seen in assignment 05.

```
# -----PRESENTATION-----#
# Main Script Body
while(True):
    print("""
    Menu of Options
    1) Enter and append data to binary file
    2) Read 1 line of existing data in binary file
    3) Read all existing data in binary file
    """)

    strChoice = str(input("Which option would you like to perform? [1-3]:  "))
    print() # line for looks
```

Figure 10. While Loop – Initial Main Menu

As the `pickle.load()` function only unpickles 1 line from the binary file, a new function has to be made in order to unpickle the other lines. This function is named: `read_ALLdata_from_file`.

An initial attempt at this function is listed below:

```
def read_ALLdata_from_file(file_name):
    """
    This function reads all rows of existing binary data from the binary file

    :param file_name: Name of binary file
    :return: Returns all rows of list_of_data which is converted from binary to text
    """
    try:
        file = open(file_name, "rb") # rb = r refers to read, b refers to binary
        for row in file_name:
            list_of_data = pickle.load(file)
            print(list_of_data)
        file.close()
    return list_of_data
```

Figure 11. New Function to Read all Lines of Binary Data in File

When running this function as a test, an exception error was encountered: EOFError.

```

Menu of Options
1) Enter and append data
2) Read existing data
3) Read all existing data

Which option would you like to perform? [1-2]: 3

[20, 'lamp']
[3, 'toast']
Traceback (most recent call last):
  File "C:\_PythonClass\Module07 - Files and Exceptions\test7.py", line 82, in <module>
    print(read_ALLdata_from_file(strFileName))
  File "C:\_PythonClass\Module07 - Files and Exceptions\test7.py", line 50, in read_ALLdata_from_file
    list_of_data = pickle.load(file)
EOFError: Ran out of input

Process finished with exit code 1

```

Figure 12. EOFError on read_ALLdata_from_file

When looking into this specific error in the Python documentation, the error description essentially describes an end of file condition [1]. This means the binary file has no more lines to load. With this information in hand, a couple of error handling blocks are built into the function in addition to some print statements as a return. This eliminates the need to print out the return of a function as the print statement is built into the return function. Only a function call will suffice. In other applications this may not be desirable, but for this purpose it is adequate. The documentation of the function was also updated to remind the user that this function has the print function built into the return statement.


```

def read_ALLdata_from_file(file_name):
    """
    This function reads all rows of existing binary data from the binary file.
    As the function automatically returns a print statement, you only need to call the function (don't print call)

    :param file_name: Name of binary file
    :return: Returns and PRINTS all rows of list_of_data which is converted from binary to text. Call function only
    """
    try:
        file = open(file_name, "rb") # rb = r refers to read, b refers to binary
        for row in file_name:
            list_of_data = pickle.load(file)
            print(list_of_data)
        except EOFError as e:
            file.close()
            return print("No more contents in file!!")
        except Exception as e:
            file.close()
            return print("unspecified error!")

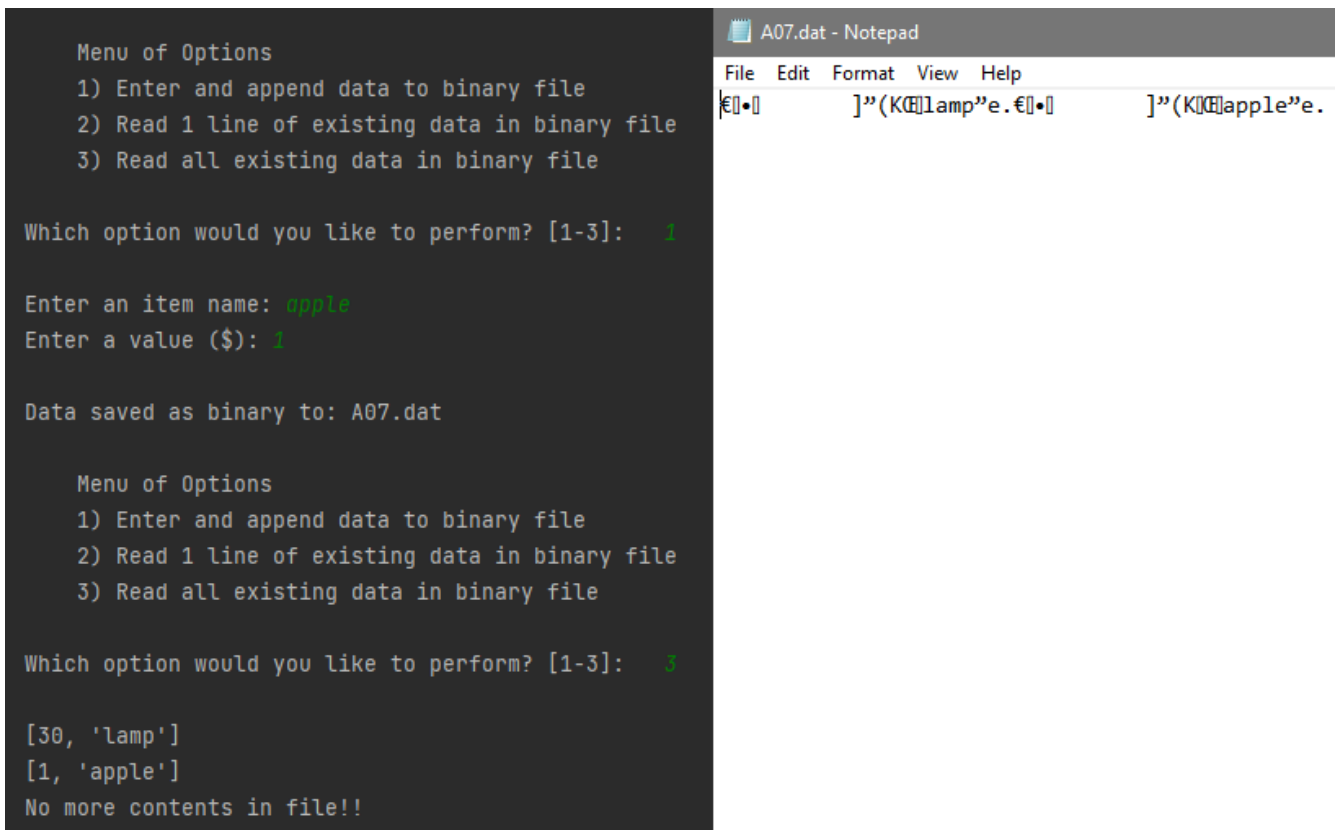
```

Figure 13. read_ALLdata_from_file Function with Error Handling of EOFError

The error handling script in demo 1, section 4.1 is then incorporated into the overall script to complete it alongside some minor quality of life additions (option to exit menu).

7.0 Verification of Script in PyCharm / Windows CMD

The script was run in PyCharm and Windows CMD to verify general functionality. The error exception functionality and pickling functionality of the script was verified in previous sections.



The image shows a PyCharm terminal window and a Notepad file named A07.dat. The terminal displays a menu of options for a binary file program. The user selects option 1 to append data, enters 'apple' as the item name and 1 as the value. The data is saved to A07.dat. The user then selects option 3 to read all existing data. The terminal shows the output: [30, 'lamp'], [1, 'apple'], and 'No more contents in file!!'. The Notepad file A07.dat contains the binary representation of the data, which appears as garbled text:]"(K  lamp"e.   ]"(K  apple"e.

```

Menu of Options
1) Enter and append data to binary file
2) Read 1 line of existing data in binary file
3) Read all existing data in binary file

Which option would you like to perform? [1-3]: 1

Enter an item name: apple
Enter a value ($): 1

Data saved as binary to: A07.dat

Menu of Options
1) Enter and append data to binary file
2) Read 1 line of existing data in binary file
3) Read all existing data in binary file

Which option would you like to perform? [1-3]: 3

[30, 'lamp']
[1, 'apple']
No more contents in file!!

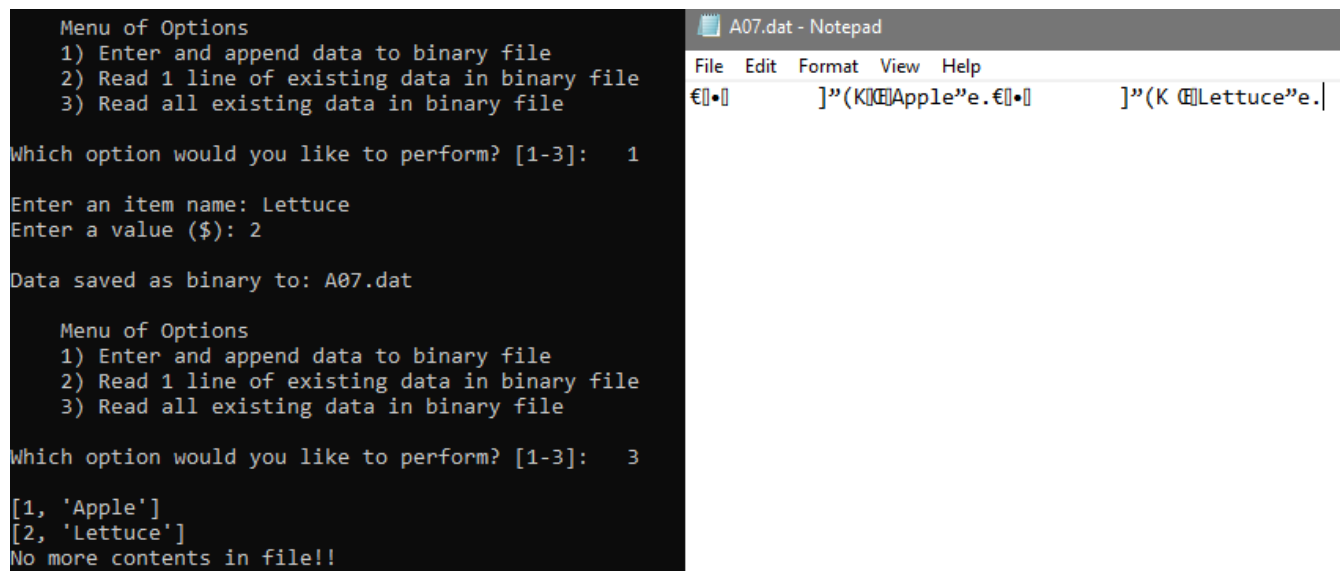
```

A07.dat - Notepad

File Edit Format View Help

   ]"(K  lamp"e.   ]"(K  apple"e.

Figure 14. PyCharm Output and Binary File



The image shows a Windows Command Prompt window and a Notepad file named A07.dat. The Command Prompt displays the same menu of options as Figure 14. The user selects option 1 to append data, enters 'Lettuce' as the item name and 2 as the value. The data is saved to A07.dat. The user then selects option 3 to read all existing data. The Command Prompt shows the output: [1, 'Apple'], [2, 'Lettuce'], and 'No more contents in file!!'. The Notepad file A07.dat contains the binary representation of the data, which appears as garbled text:    ]"(K  Apple"e.   ]"(K  Lettuce"e.

```

Menu of Options
1) Enter and append data to binary file
2) Read 1 line of existing data in binary file
3) Read all existing data in binary file

Which option would you like to perform? [1-3]: 1

Enter an item name: Lettuce
Enter a value ($): 2

Data saved as binary to: A07.dat

Menu of Options
1) Enter and append data to binary file
2) Read 1 line of existing data in binary file
3) Read all existing data in binary file

Which option would you like to perform? [1-3]: 3

[1, 'Apple']
[2, 'Lettuce']
No more contents in file!!

```

A07.dat - Notepad

File Edit Format View Help

   ]"(K  Apple"e.   ]"(K  Lettuce"e.

Figure 15. Windows CMD Output and Binary File

8.0 Research of Error Handling and Pickling

When conducting research of error handling and pickling, most search results on the internet resulted in independent websites / personal blogs. These tended to be easier to understand for beginner level programmers (such as myself). The official Python documentation was useful only for looking up certain exceptions and how they are organized. The official documentation is not as beginner friendly to read.

9.0 Summary

Research was conducted to better understand error handling and pickling in Python. Both of these techniques are used widely to create better code. When it came to understanding these concepts, it was found that independent websites and personal blogs were easier to understand than official Python literature.

In addition, a script was created based on assignment 05-06, Lab 7-1 to showcase incorporation of error handling and pickling. This script called for the user's input of an item and the item's value, placed it in a list, and then appended it to a binary file (pickling) where it can then be converted back to text to read (unpickling). One of the challenges of the script was getting around the `pickle.load()` function as it only read 1 line. Understanding error handling was crucial to solving this issue in the script.

Although the script had only a handful of options, an improvement could be made to incorporate both error handling and pickling to more options (and functions) such as removing specific lines and having write functionality to overwrite the contents.

List of Works Cited

- [1] “Built in Exceptions,” docs.python.org. <https://docs.python.org/3/library/exceptions.html#builtin-exceptions> (accessed Nov 26, 2022).
- [2] “8. Errors and Exceptions,” docs.python.org. <https://docs.python.org/3/tutorial/errors.html> (accessed Nov 28, 2022)
- [3] S. Van de Klundert. “Python Exceptions: An Introduction,” realpython.com. <https://realpython.com/python-exceptions/> (accessed Nov 29, 2022)
- [4] R. Root. (2022). IT FDN 110 A Python Programming: Module 7 [Word document].
- [5] “Python Pickling,” tutorialspoint.com. <https://www.tutorialspoint.com/python-pickling> (accessed Nov 29, 2022)