

Barbara Spadavecchia

March 1, 2022

IT FTN 110 A

Assignment07

<https://github.com/Barb4000/IntroToProg-Python-Mod7>

Research on Exception Handling and Pickling in Python

Introduction

This paper discusses the benefits of using structured error handling and how to use exception handling in python. This assignment helped build a better understanding of python's built-in error handling and how it could be customized to provide more user-friendly error messages. This assignment also covered how to save and read data in binary through a python process of pickling and unpickling. After research on exception handling and pickling, these processes were demonstrated in a script.

Exception Handling in Python

An exception is any event that occurs during the execution of the program and disrupts the program's instructions. It is an error that needs to be handled immediately or the script will quit. In python, an exception is an object that will give information on the error type, state of the program when the error occurred, and an error message that describes the event. Syntax errors, known as parsing errors, occur when the proper structure or syntax of a language is not followed. Syntax errors are not exceptions but an error detected during execution is called an exception. Examples of standard exceptions are: `FileNotFoundException`, `ImportError`, `RuntimeError`, `NameError`, and `TypeError`.

Benefits of Using Exception

Using the built-in exceptions gives a standardized way to manage errors. The ability to create a customized exception helps give a more precise name to the error, which helps in debugging the error event. Code is cleaner when using exceptions because error-handling code is separated from regular processing code and follow the best practice of separation of concerns. Error types can be classified by type.

Try and Except Block to Handling Exceptions

Risky code that can raise an exception should be defined in a try block and should be handled in the except block. It is a good practice to specify an exact exception. Multiple numbers of except blocks can follow a try block but only one will be executed when an exception occurs.

Raising an Exception

The raise statement is useful when an exception needs to be raised to the caller program. Steps to raise an exception are create an exception of the appropriate type, pass the data while raising an exception, execute a raise statement by providing the exception class.

Resources Used to Explore Exception Handling

https://www.w3schools.com/python/gloss_python_error_handling.asp is a website that gives a simple explanation of exception handling.

https://www.tutorialspoint.com/python/python_exceptions.htm This website goes over the two major ways that python manages any unexpected errors: exception handling and assertions. This site provides a list of standard exceptions and their descriptions.

<https://realpython.com/lessons/raising-an-exception/> This website breaks down exception handling into a tutorial with videos. The downside is that to watch all of them you would have to join.

<https://pynative.com/python-exceptions/> This website is written and maintained by a python developer who is also a python enthusiast. It provides detailed examples of how to use python to manage exceptions using the try, except, and finally statements.

<https://www.python.org/doc/essays/stdexceptions>

Python.org provided helpful examples of the exceptions and was a good reference.

What is Pickling

Pickle is used to convert an object in memory to a byte stream that can be stored on a disk. This character stream can be retrieved and de-serialized back to a python object. It is a way to back up important objects and pass objects between scripts. Pickling is useful when you need your data to persist. For example, if you are working with time consuming learning algorithms, data will load far faster by serializing the data by pickling it. This process was applied by importing pickle and converting a text file into a binary file. Pickle is not recommended to use with data across different programming languages. Always make sure that the source is trusted before unpickling or potentially malicious code might be executed. The following data types can be pickled: Booleans, integers, floats, complex numbers, strings, tuples, lists, sets and dictionaries.

Resources Used to Explore Pickling

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

This website was helpful at giving brief examples of how to pickle and a brief explanation of the topic.

<https://www.geeksforgeeks.org/understanding-python-pickling-example/>

This website provided another example of pickling data loaded into a dictionary.

<https://ianlondon.github.io/blog/pickling-basics/>

Blog site giving the basics of pickling.

<https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>

This site gave short examples of pickling and good reasons to use pickling to store python objects.

Demo 1: Pickling and Unpickling

The basic steps are to first import pickle to use it, then input contact information to a list, which is a python object. Next, open a file to write bytes in Python3+, then use pickle.dump() to put the list into the opened file, then close the file.

```
#####  
# Title: Assignment07 Contact Information  
# Dev: BSpadavecchia  
# Date: February 28, 2022  
# Change log: (Who,When,What)  
# BSpadavecchia, 02-28-2022, Created Script  
#####  
# Description: Demonstration of pickling and error handling in python.  
  
# Pickling demo  
  
import pickle  
  
# Collect contact information to pickle  
contact_last_name = str(input("Enter last name of contact: "))  
contact_first_name = str(input("Enter first name of contact "))  
contact_email = str(input("Enter contact email "))  
contact_cell = str(input("Enter cell number of contact "))  
contact_lst = [contact_last_name, contact_first_name, contact_email,  
contact_cell]  
print(contact_lst)  
  
# Save data to a text file using pickle.dump  
print("Saving contact data to a file using pickle.dump")  
myfile = open("contacts.txt", "ab")  
pickle.dump(contact_lst, myfile)  
myfile.close()  
print("Contact data pickled and saved")  
  
# Read and display saved data using pickle.load  
print("Reading contact data to a list from a binary file using pickle.load")  
myfile = open("contacts.txt", "rb")  
mydata = pickle.load(myfile)  
print(mydata)  
myfile.close()
```

Figure 1 Demo 1 Code to pickle to a file


```

C:\Users\bspad>python.exe "C:\_PythonClass\Assignment07\Assignment07.py"
Enter last name of contact: Smith
Enter first name of contact Frank
Enter contact email SmithF@mail.com
Enter cell number of contact 1112224444
['Smith', 'Frank', 'SmithF@mail.com', '1112224444']
Saving contact data to a file using pickle.dump
Contact data pickled and saved
Reading contact data to a list from a binary file using pickle.load
['Smith', 'Frank', 'SmithF@mail.com', '1112224444']
Press Enter to continue to Error Handling Demo

```

Figure 2 Running Demo 1 from Command Prompt

Demo 1 Pickling data

 contacts.txt - Notepad

File Edit Format View Help

```

[•4      ]”(“Smith”“Frank”“SmithF@mail.com”“
1112224444”e.

```

Figure 3 Binary file Contacts.txt

```

#Error Handling Creating Custom Classes
print("Press Enter to re-run the program with error handling")
print()

class AlphaError(Exception):
    """ Error in not using alphabetical letters for name """
    def __str__(self):
        return 'Please use alphabetical letter for name fields'
class EntryError(Exception):
    def __str__(self):
        return "Entry not accepted: Please enter at least one character in
contact_last_name"
class NumError(Exception):
    def __str__(self):
        return "Please use only numbers in cell_number field"
class CellNumError(Exception):
    def __str__(self):
        return "Please use 10 numbers in cell_number field"

try:
    contact_last_name = input("Enter your contact's last name: ")
    if contact_last_name.isnumeric():
        raise AlphaError()
    elif len(contact_last_name) == 0:
        raise EntryError()
    try:
        cell_number = input("Enter your contact's cell phone number: ")
        if cell_number.isalpha():
            raise NumError()
        elif len(cell_number) != 10:

```

```

        raise CellNumError()
    else:
        # Save data to a text file using pickle.dump
        new_contact_lst = [contact_last_name, contact_first_name,
contact_email, cell_number]
        print("\n" "Entry complete. Preparing to save to contacts.txt")
        myfile = open("contacts.txt", "ab")
        print("Saving contact data to a file using pickle.dump")
        pickle.dump(new_contact_lst, myfile)
        myfile.close()
        print("Contact data pickled and saved")
    # Read and display saved data using pickle.load
    print("Reading contact data to a list from a binary file using
pickle.load")
    myfile = open("contacts.txt", "rb")
    mydata = pickle.load(myfile)
    print(mydata)
    myfile.close()
    print("Program for Assignment07 is over")
except Exception as e:
    print("\n" + "Error: " + "\n")
    print(e)
except Exception as e:
    print("\n" + "Error: " + "\n")
    print(e, e.__doc__, e.__str__, type(e), sep="\n")

```

Figure 4 Error Handling showing try except blocks