Hwai-Jiang Jong

May 24, 2021

Foundations of Programming: Python

Assignment 07

<https://github.com/uwp-h2021/IntroToProg-Python-Mod07>

Exception Handling and Pickling

# Introduction

This report documents my research results on exception handling and pickling. It also describes a Python script I created to demonstrate my learning. It contains five main sections (excluding this introduction):

1. Exception Handling Research;
2. Pickling Research;
3. Exception and Pickling Script;
4. GitHub Webpage
5. Summary

# The Exception Handling

The following website was what I read and found very informative.

<https://realpython.com/python-exceptions/> (external site): This website covers many details about the exception handling in Python. The examples listed in each of the clauses facilitate clear understanding of the way to handle exceptions.

It starts with two simple ways to handle the exception by raising an exception and asserting an exception (Figure 1). Both ways result in the program being halted if an exception is encountered.

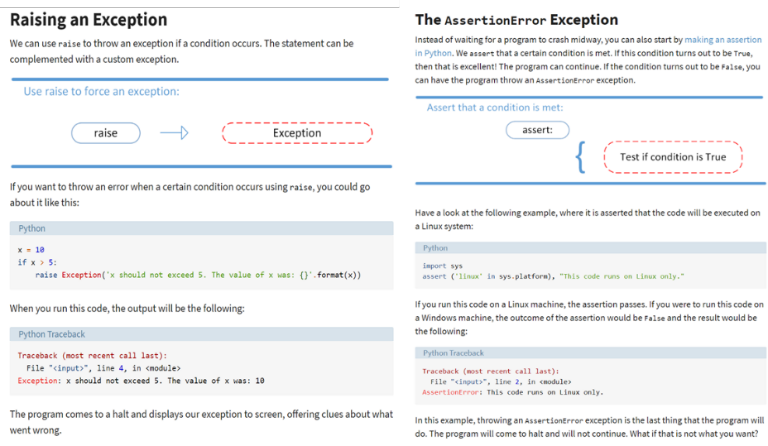


Figure Raise an Exception and Assert an Exception: (Source: <https://realpython.com/python-exceptions/> )

The *try-except* block (Figure 2) is introduced in a progressive way so the readers learn how to capture exceptions in Python codes. There are elegant ways to handle exception using various options to print the messages, such as built-in exception types or programmer defined assertion exception.

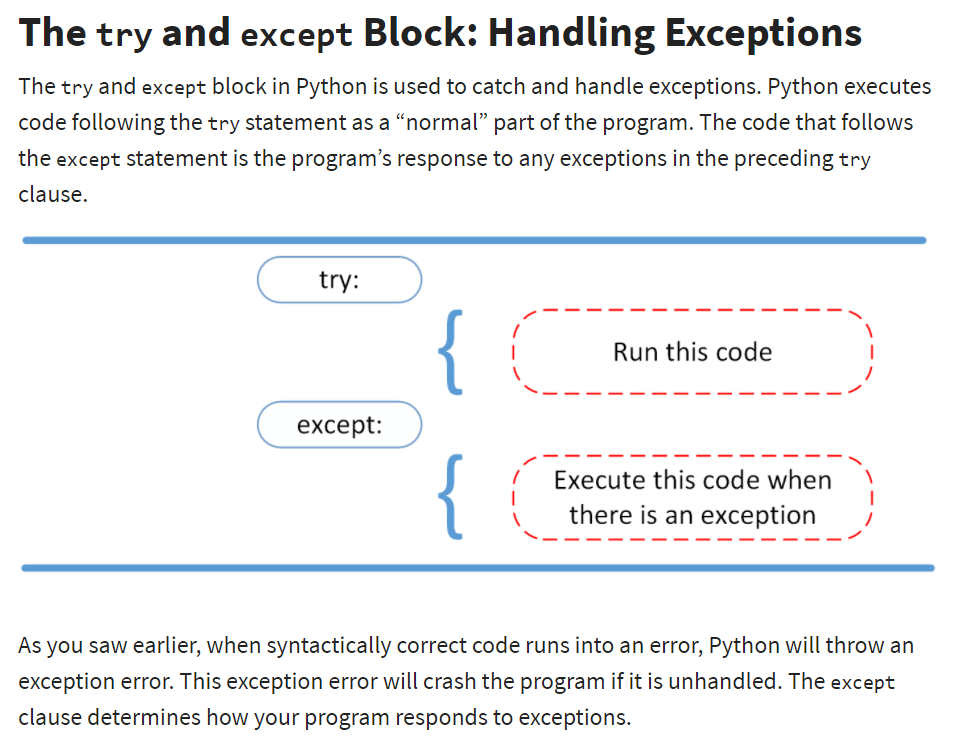


Figure The try-except Block: (Source: <https://realpython.com/python-exceptions/>)

By adding the *else* clause after the *try-except* block, the script can execute a block of code in the absence of the exception, as seen in Figure 3.

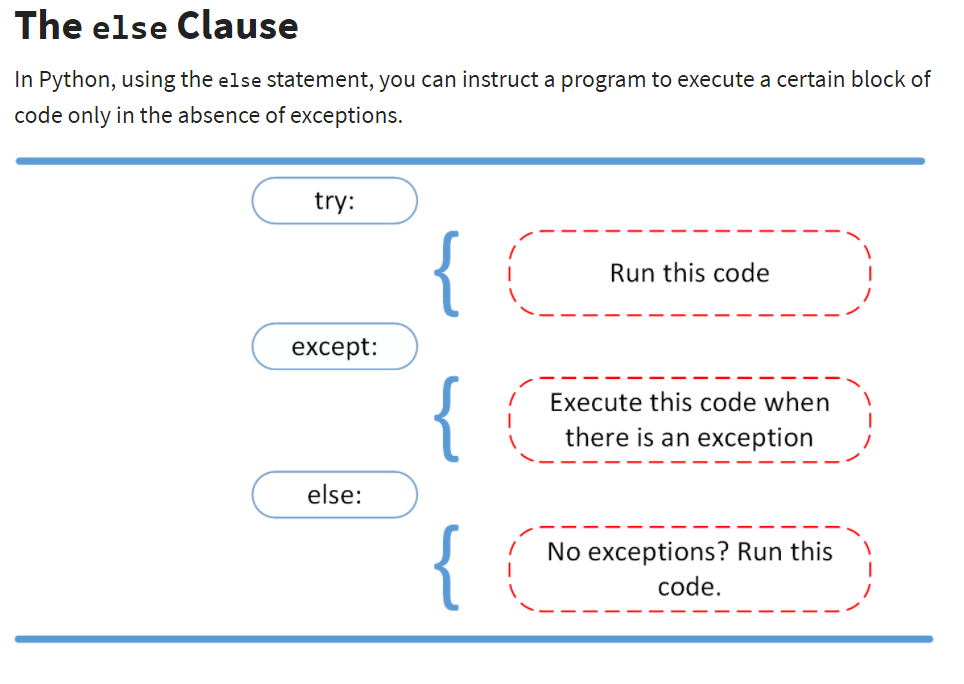


Figure The else Clause (Source: <https://realpython.com/python-exceptions/> )

We can even *try* in the *else* clause to catch any exception that is in the absence of the other exception. For example, Figure 4 shows that in the absence of the *AssertionError* regarding the system, the code also captures the file opening status using a separate *try-except* block within the *else* clause.

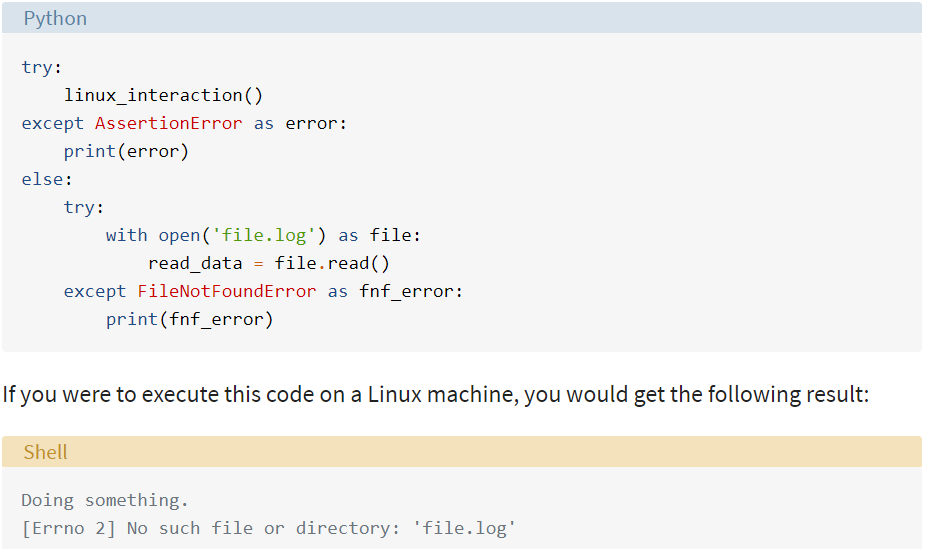


Figure Try-except block within an else Clause (Source: <https://realpython.com/python-exceptions/>)

Finally, as seen in Figure 5, the *finally* clause allows the block of code to be executed regardless of the exception conditions that go before it.

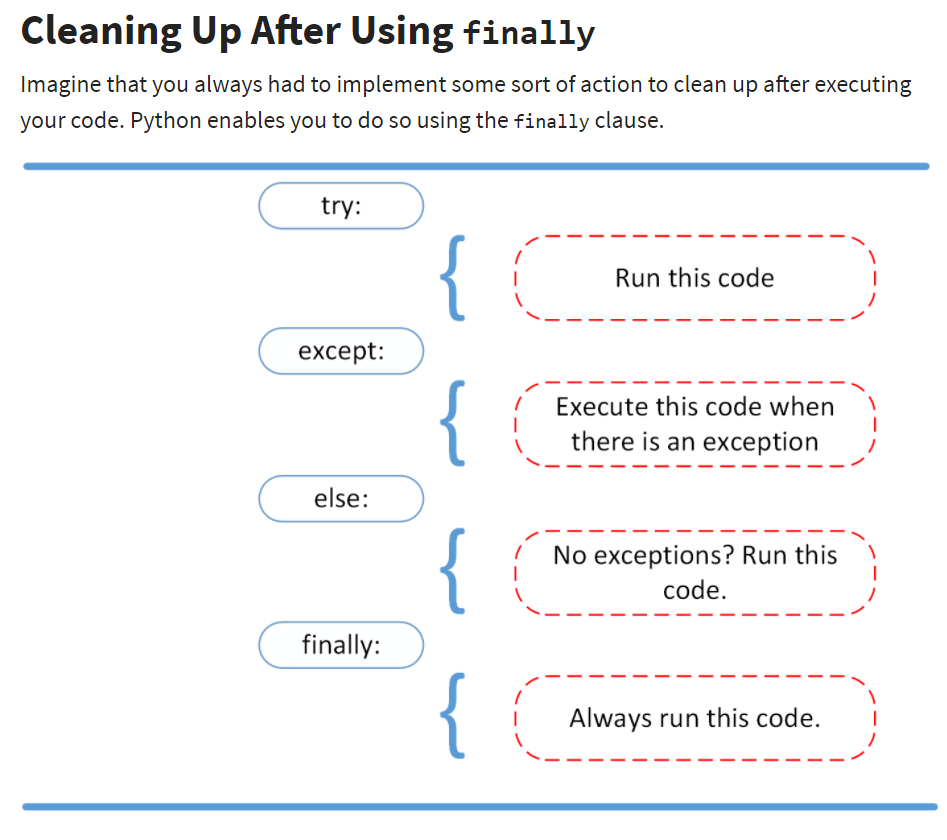


Figure Cleaning up After Using the finally Clause (Source: <https://realpython.com/python-exceptions/>)

# Pickling and Unpickling

<https://realpython.com/python-pickle-module/> (external site): This website discusses serialization in Python, and why and in what situation should we do it. I learned three serialization modules in the standard library, in which the *pickle* module is used in most of the situation when we don’t need the human readable format or standard interoperable format in serialization of objects. This website is good in that the subject is presented in a progressive way, and the attached examples for the *pickle* module are convenient for me to run it on my machine to solidify the concept.

As shown in Figure 6, four methods in *pickle* module are discussed: two methods are to *pickle* an object into a string, and the other two into a file.

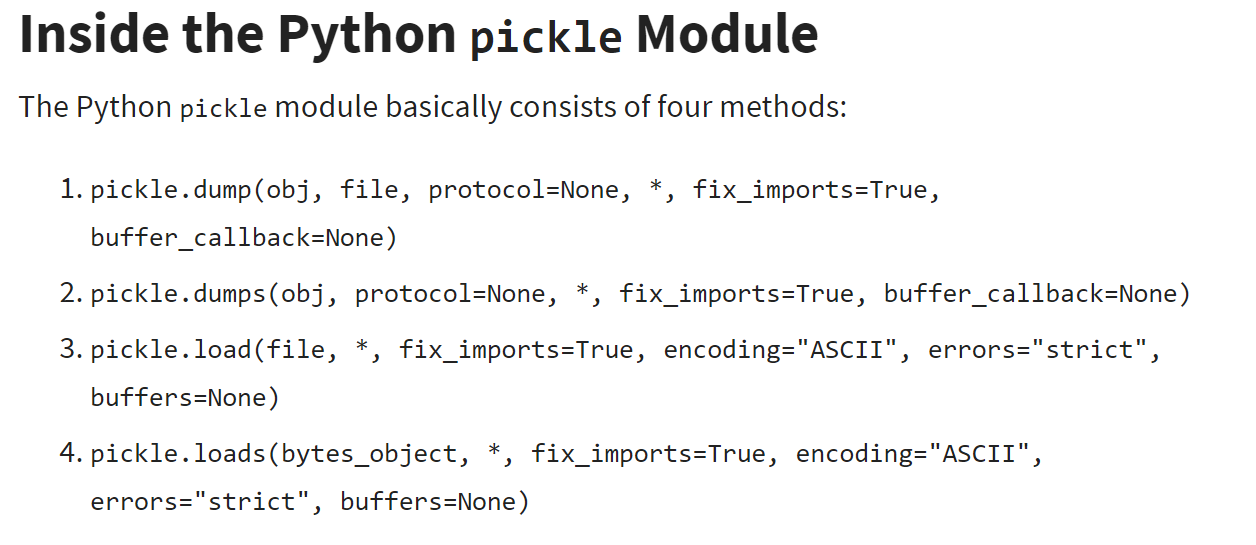


Figure The dump(), dumps(), load(), and loads() Methods in Pickle Module (Source: <https://realpython.com/python-pickle-module/>)

An example in Figure 7 demonstrates how *pickle.dumps()* and *pickle.loads()* are used to pickle and unpickle a *class* object, and how the pickling preserves the object’s attribute even after it is modified after pickling.



Figure pickle.dumps() and pickle.loads() Methods for Pickling and Unpickling a Class Object (Source: <https://realpython.com/python-pickle-module/>)

Finally, the website discusses a way to compress a pickled object by using the *compress()* method in *bz2* module, as seen in Figure 8.

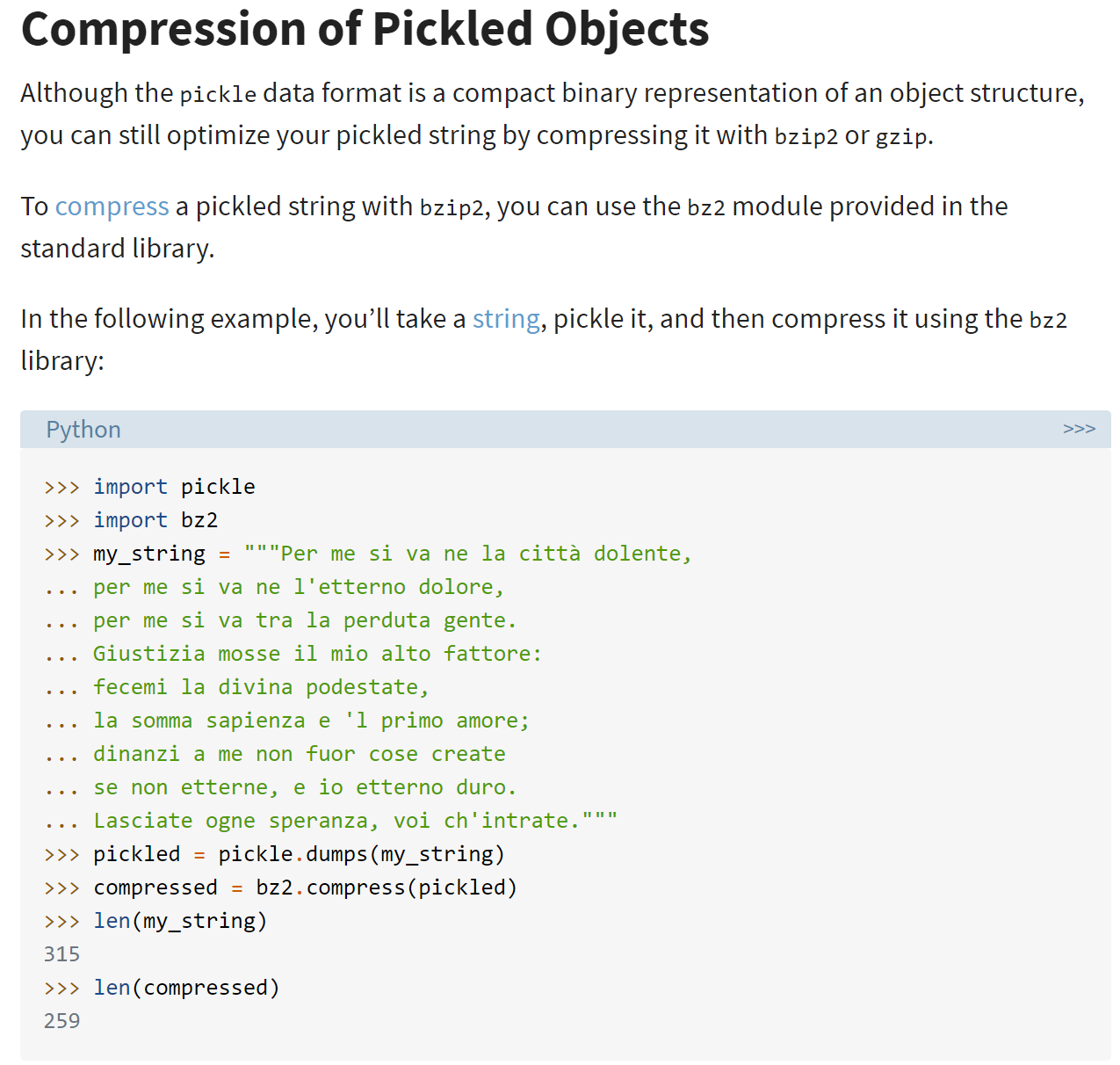


Figure Compression of Pickled Objects (Source: <https://realpython.com/python-pickle-module/>)

# Writing the Exception Handling and Pickling Script

## Script Planning

In this script, I am demonstrating exception handling and data pickling in Python. The following are the steps I took and have implemented in the code.

1. Create a text data file that contains multiple lines of string. The text data file name is “*DataFile.txt*” stored in the hard drive, and its content is shown in Figure 9.
2. Ask the user to enter the text file name for pickling. Handle the exception when the entered file name does not exist using *try-except* block.
3. Read the data from the text file and store them into a list of strings.
4. Write the list of strings into a binary file using *pickle.dump() method.*
5. Read the pickled data from the binary file back into a list of strings using *pickle.load() method.*
6. Print the list of strings to the user.

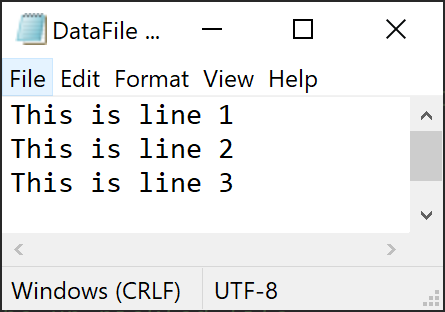


Figure The “DataFile.txt” With Its Content

## Writing the Script in PyCharm

I created a new project folder “C:\\_PythonClass\Assignment07\”, in which the script “Assignment07.py” was created and tested for exception handling and pickling.

Following the steps in the planning, the script was completed with its various sections described in detail as follows.

1. Change History: Figure 10 shows the history in creating the script.

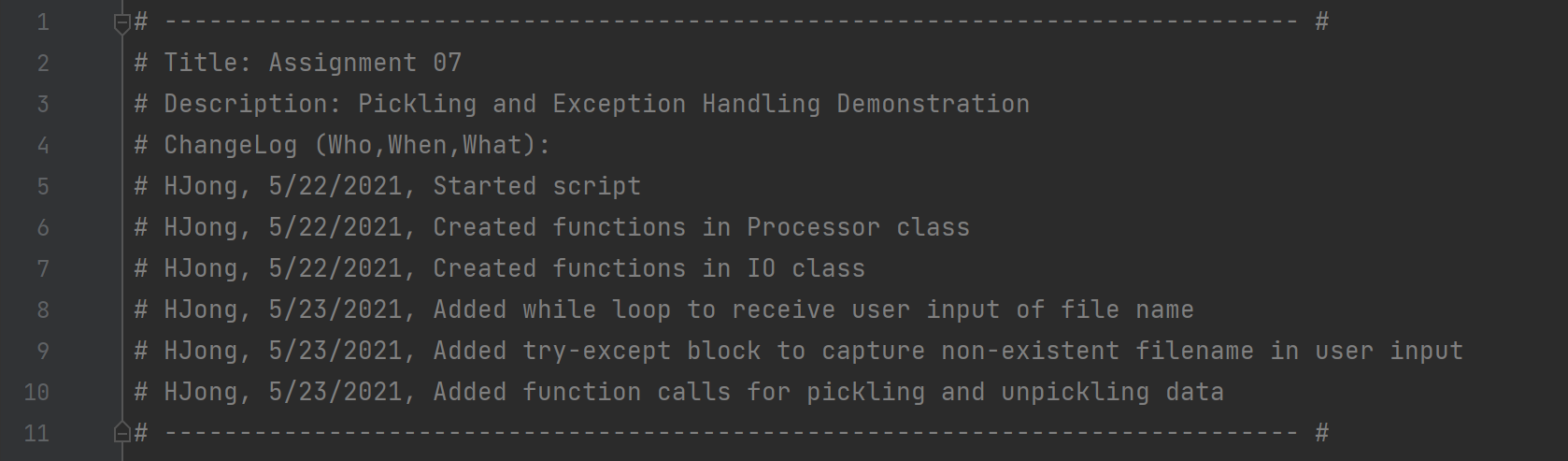


Figure Change Log of the Script

1. Data definition: Figure 11 shows the variables I used in the script to perform exception handling and pickling.

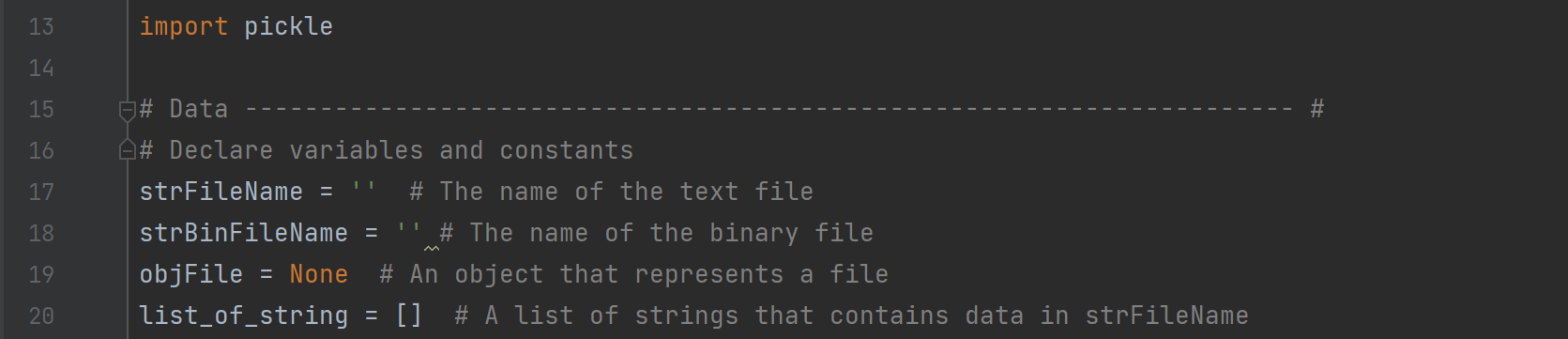


Figure Data Declaration Section of the Script

1. Data Processing – This section contains a class *Processor* which consists of *read\_data\_from\_file(), pickle\_it(), and unpickle\_it()* functions. See Figure 12.



Figure The Class “Processor” for Processing Data

1. Presentation – This section contains the class *IO* which consists of *get\_user\_filename() and display\_data()* functions, as shown in Figure 13.

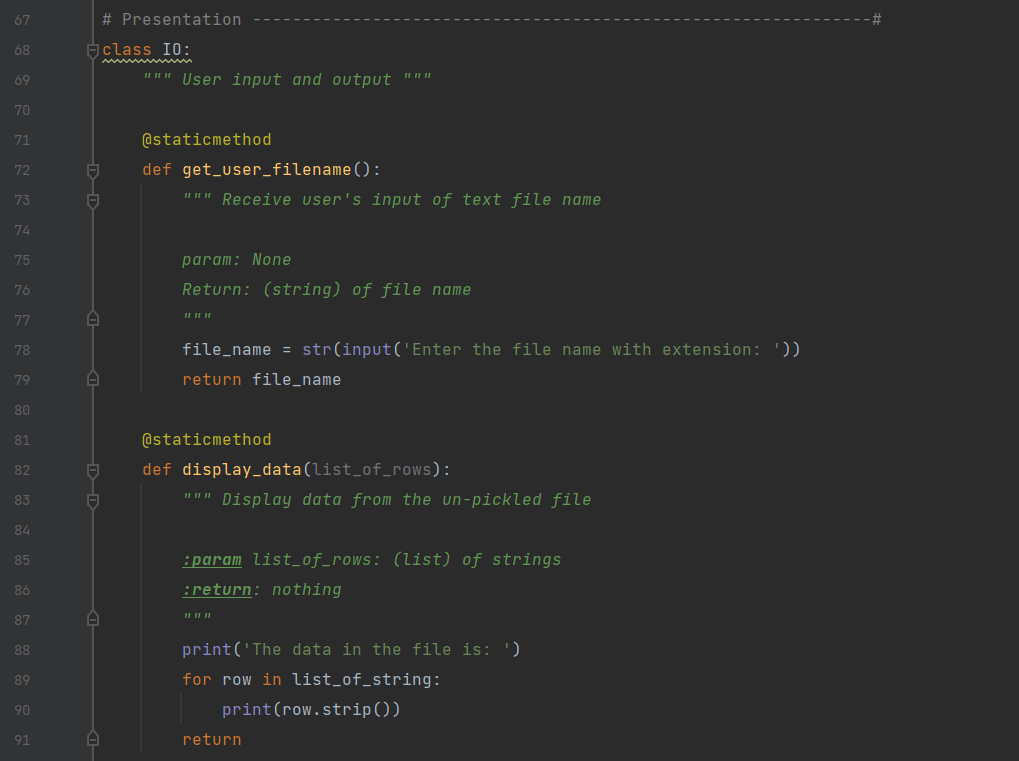


Figure The Class “IO” for Presentation

1. Main program – The main program consists of exception handling and pickling subsections. I fulfilled the code to receive user input of text data file name, read data from the text file, write the data into a binary file, and then read the binary file back to a list of strings. At the end, the list is printed to show the data has been processed correctly. These actions were done by calling functions in the *Processor* and *IO* classes. Figure 14 shows the implementation in the main program.

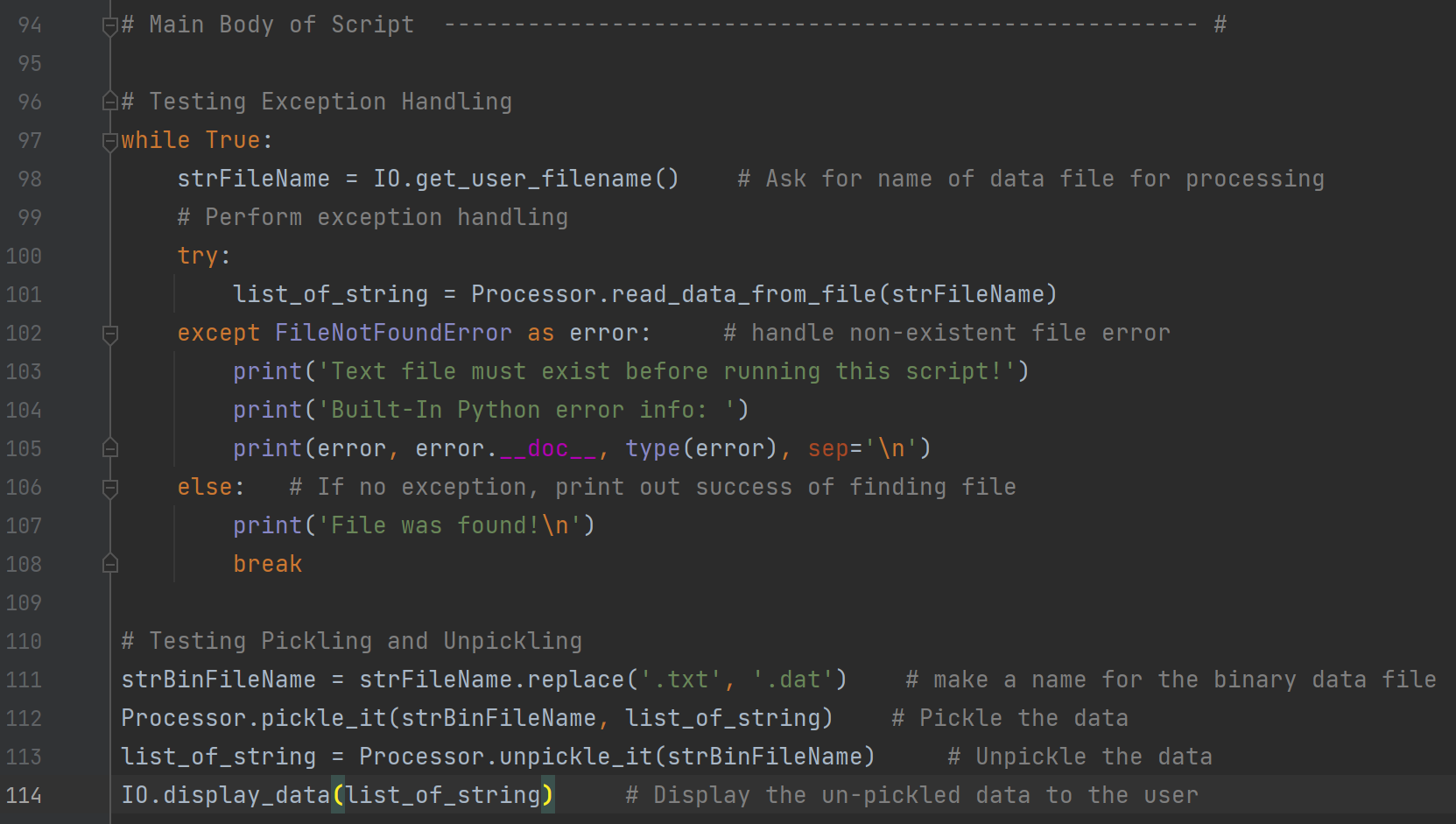


Figure Main Program that Processes User Choice Options

## Executing the Script in PyCharm

In this section, I am demonstrating the script in performing the exception handling and pickling actions. I tested the exception handling by entering a non-existing file name, then an existing file name without extension, both of which were handled by putting out the error message. The script kept asking the user for an existing text file name until no exception was captured. The data from the original text file “*DataFile.txt*” were read, written into a binary file, read back into memory, and finally displayed to the user to verify the process was done correctly. The binary file name “*DataFile.dat*” was the same name as the original text file except that its extension was “*.dat*”. The run results are shown in Figure 15.

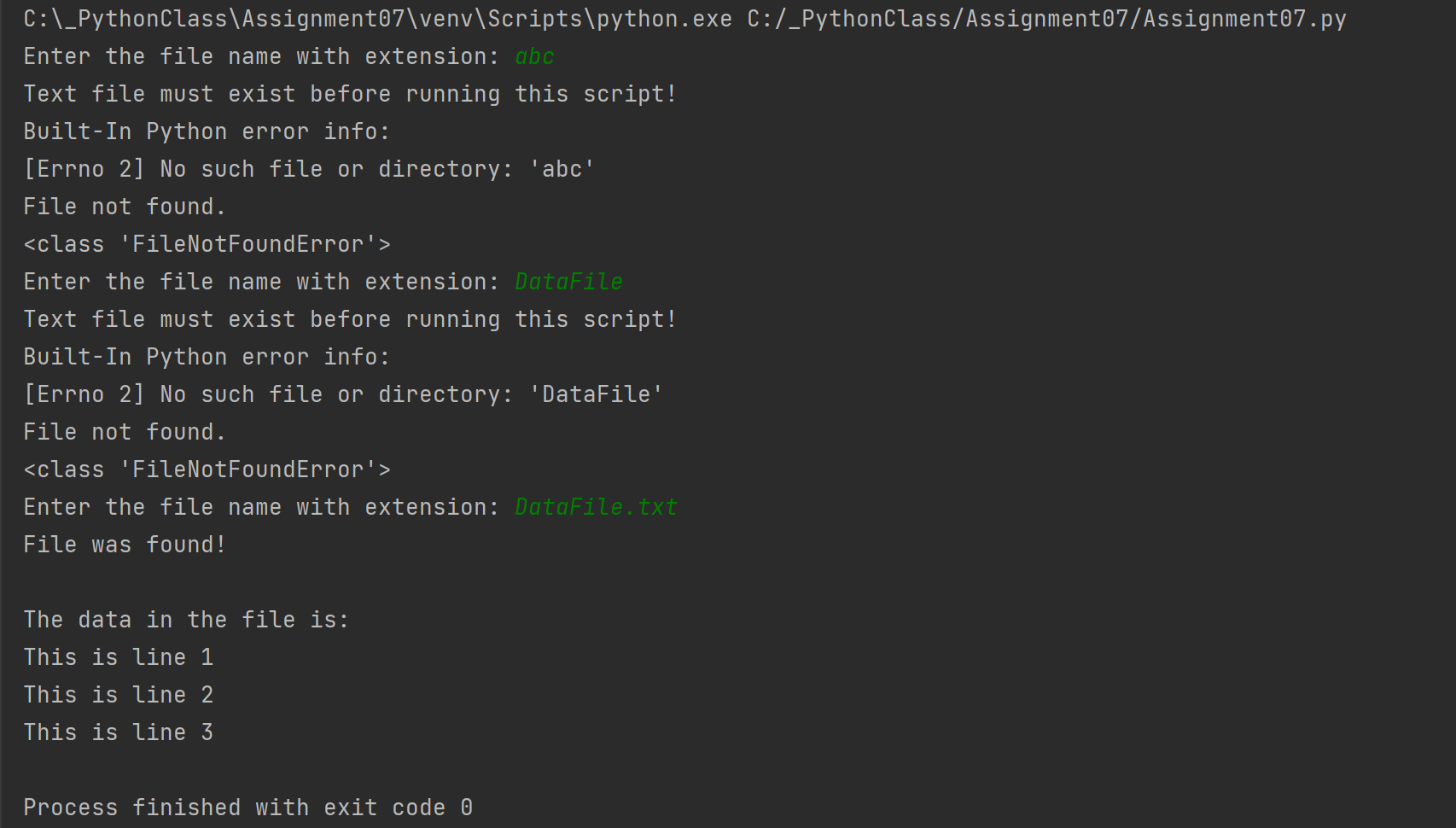


Figure Run Results in PyCharm

## Checking the Binary File After PyCharm Run

Finally, I opened “DataFile.dat” to verify the binary data were created as intended, which is shown in Figure 16.

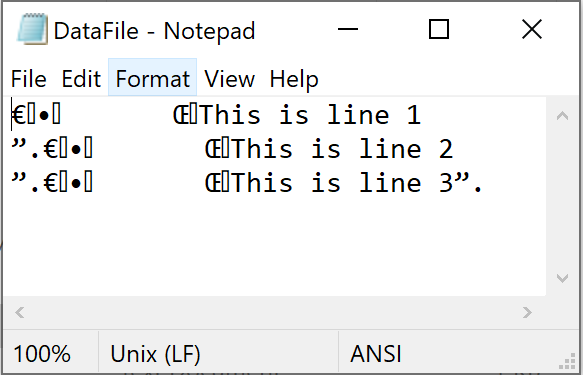


Figure Binary Data in “DataFile.dat” After the Run in PyCharm

## Executing the Script in Windows Command Prompt

I went to “C:\\_PythonClass\Assignment07\” and run Assignment07.py in Windows command prompt. Same steps as the PyCharm run were followed, and the results are shown in the screenshot in Figure 17.

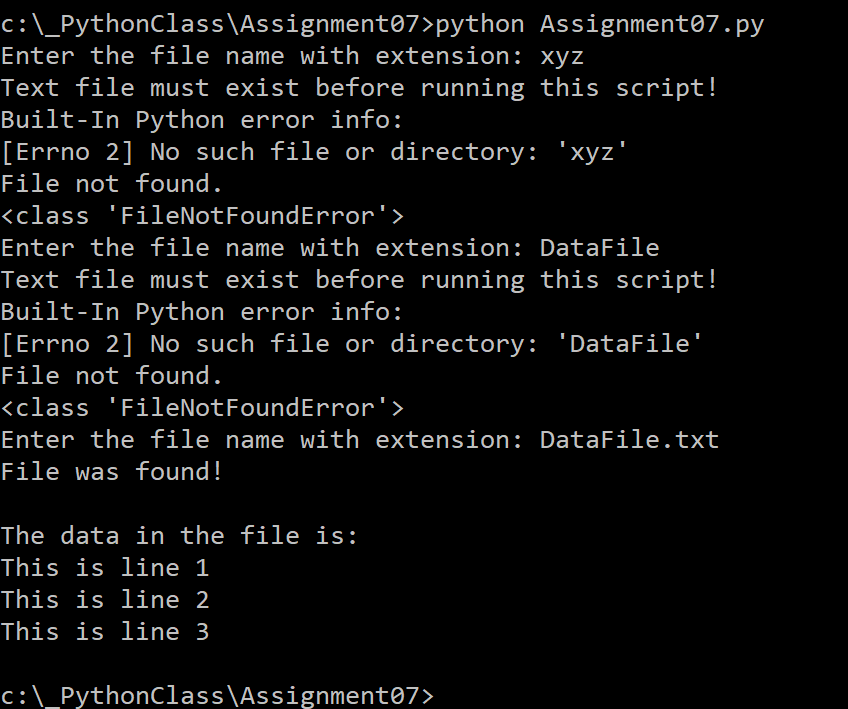


Figure Run Results in Windows Command Prompt

## Checking the Binary File After Windows Command Run

After the Windows command run, I checked the binary data in “DataFile.dat” in the working folder. As shown in Figure 18, the binary data file showed the pickling and unpickling in the script were validated in both PyCharm and Windows command.

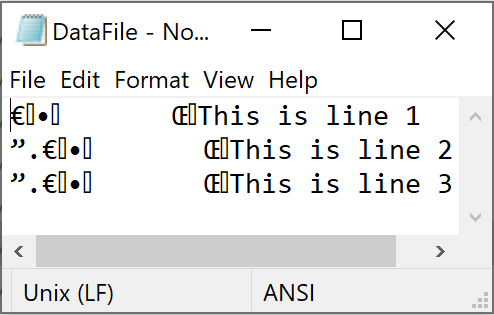


Figure Binary Data in “DataFile.dat” After the Run in Windows Command

# GitHub Webpage

The webpage was created with Assignment07 repository “IntroToProg-Python-Mod07”. A sample portion of the *index.md* is shown in Figure 19.

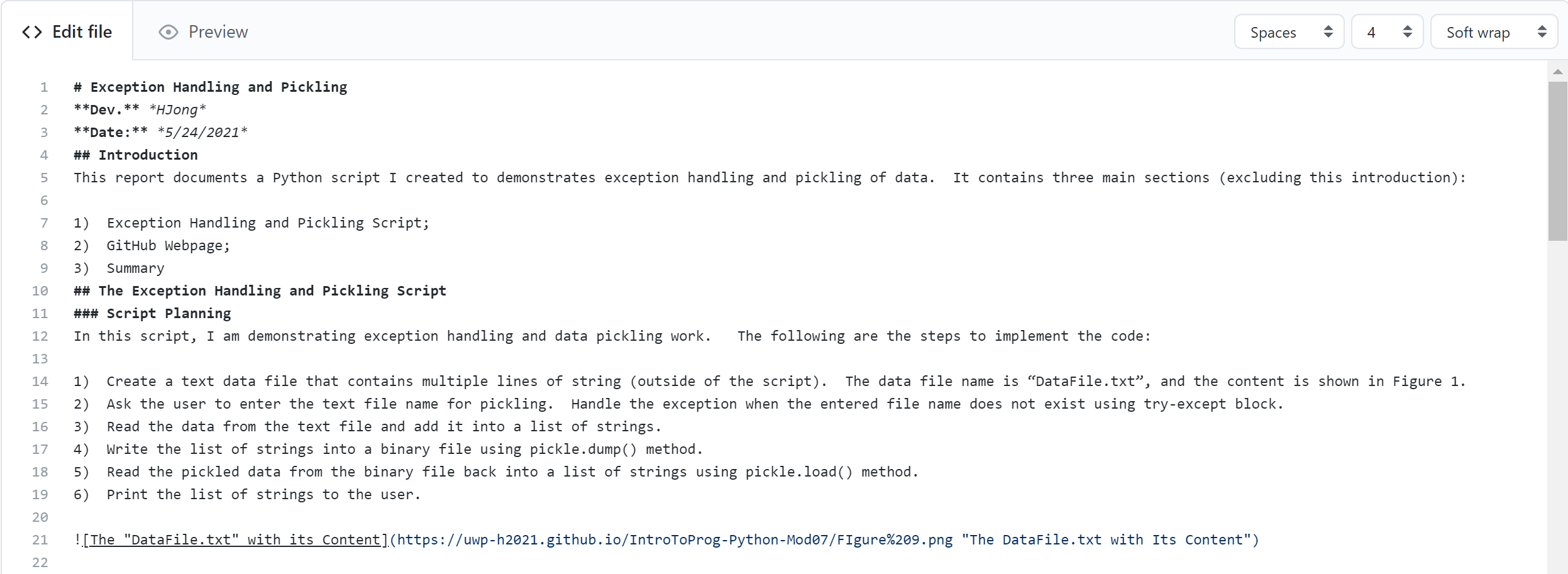


Figure A Sample of index.md in the /docs/ folder

The finished GitHub webpage is shown in Figure 20.

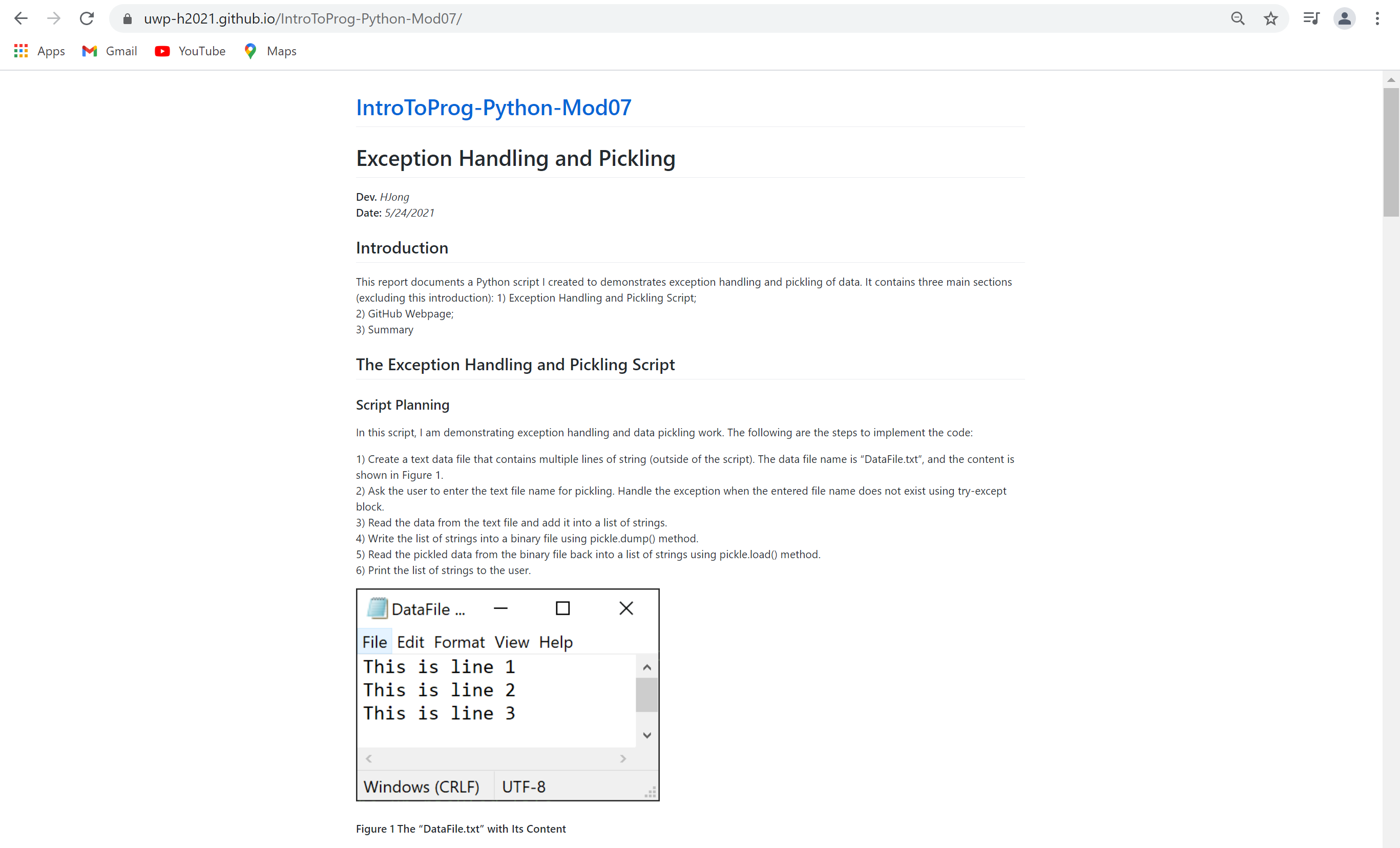


Figure GitHub Webpage for IntroToProg-Python-Mod07

# Summary

This report summarizes my research on error handling and pickling of data, together with a script that demonstrates how to handle errors and pickle the data. The GitHub webpage was created to publish the report on the internet using “*index.md” in …/docs/* folder in the repository. I expect more learning of the Markdown language in future assignments.