Jessica Carnes

November 30, 2020

Foundations of Programming: Python

Assignment07

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

Structured Error Handling and Pickling

# Introduction

To demonstrate structured error handling and pickling I have created a Python script that collects and saves blood pressure readings from a user. This script requires the user to enter integer values for both the systolic and diastolic blood pressure readings, otherwise it fails with an exception. The values are then stored in a dictionary and the dictionary in a list which is saved by pickling to a binary file.

# Structured Error Handling

Structured error handling allows a programmer to provide more descriptive feedback to a user when something goes wrong with a script. It also gives the script a way to continue to run even if it runs into an error. This is helpful whenever user input might cause a program to fail. A few examples of when structured error handling may be useful are when reading from a file if the file is not present in the expected path, dividing by zero, or an index error where the program tries to access an index that is out of range of the length of the object.

A good resource for learning about structured error handling in Python is the Tutorials Point, Python Exceptions page (<https://www.tutorialspoint.com/python/python_exceptions.htm>, 2020) (external site). This tutorial provides a comprehensive list of types of exceptions that could occur then describes how using a try and except block can help mitigate issues. In this tutorial, they define an exception as follows:

“An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.”

To overcome exceptions, programmers should utilize try and except blocks whenever they foresee an error might arise. In the blood pressure script, structured error handling is used when a user enters their blood pressure to ensure they are only entering integer values as well as when opening the binary file in read mode to ensure the file exists.

# Pickling Data

Pickling allows a user to store complex type of data, such as lists or dictionaries, to a file. The file must be stored in a binary file (.bat extension) instead of a text file. The process of pickling serializes the data as explained in the following tutorial from Data Camp (<https://www.datacamp.com/community/tutorials/pickle-python-tutorial>, 2020) (external site):

Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.

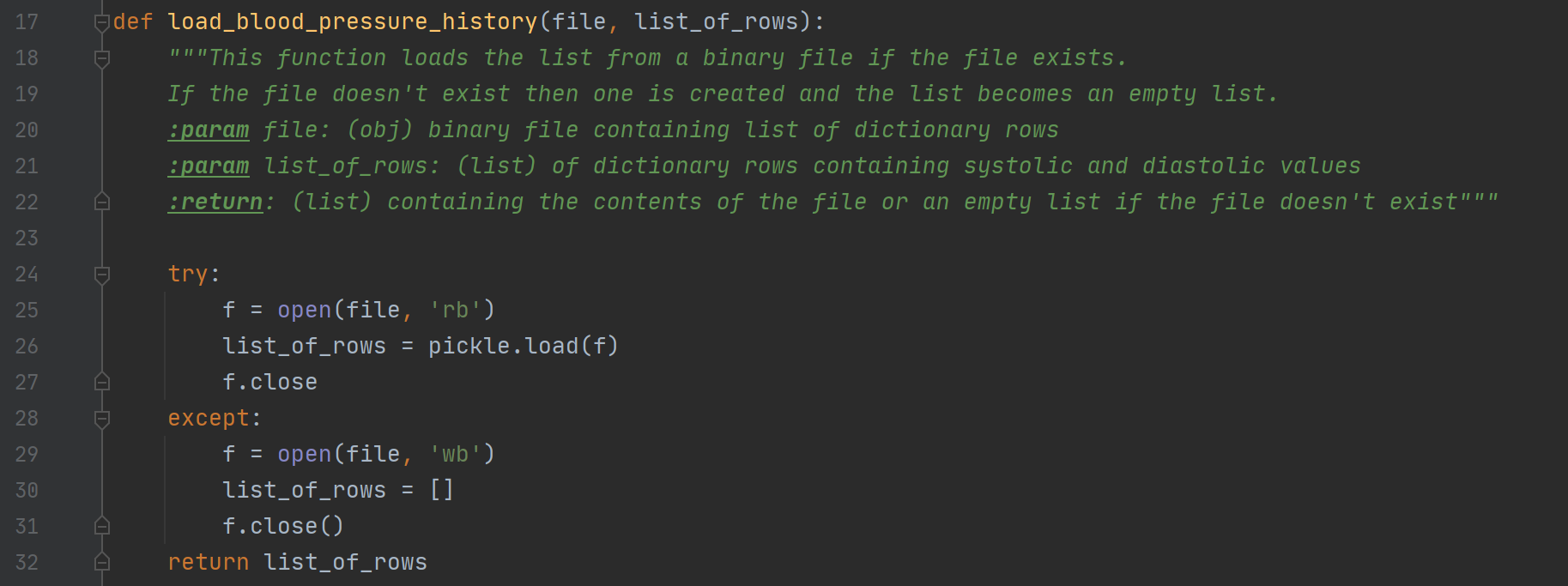
Writing to a binary file in Python is similar to the process of writing to a text file however the separate Pickle module must be imported to do so. The modes for reading, writing, and appending binary files are ‘rb’, ‘wb’, and ‘ab’ respectively. When reading to a file the command is pickle.load() and to write or append to the file the command is pickle.dump().

Another helpful tutorial on pickling was this medium post (<https://medium.com/swlh/pickling-in-python-ac3c7a045ae5>, 2020) (external site). I appreciate that the author shared how pickling can be especially useful for storing trained data in machine learning as training can take a significant amount of time.

As a note of caution, several tutorials including the medium post above, caution against opening binary files from unknown sources as this could leave your computer vulnerable for an attack.

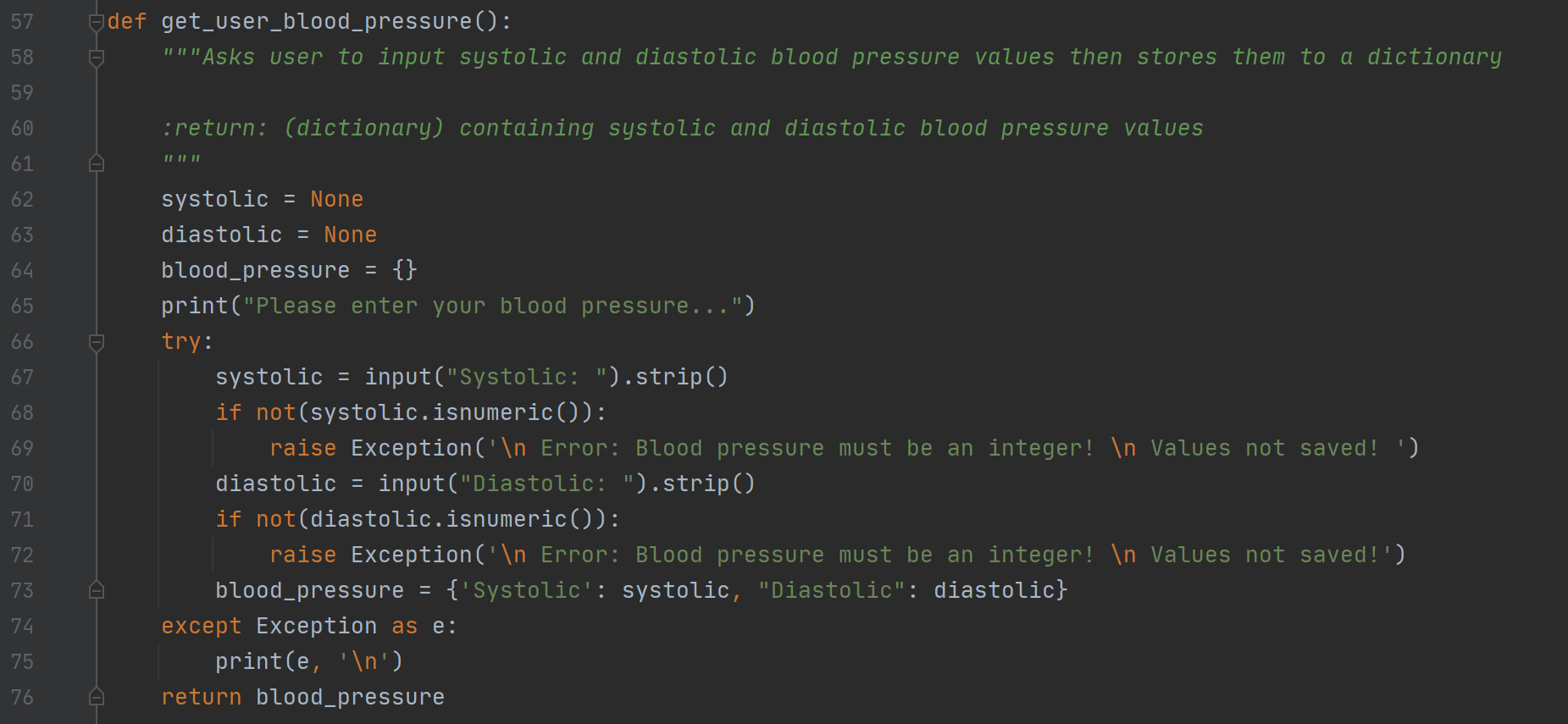
# Error Handling When Opening a Binary File

The blood pressure script uses the function load\_blood\_pressure\_history() to load the data that is currently in the blood\_pressure.dat binary file to a list. This is done using a try and except block as it is initially being opened in read mode, therefore if the file doesn’t exist an error will occur. As shown in Figure 1, the function attempts to open the file and if it is successful the data in the file is loaded to a list in python. If there is an error, then that means the file does not exist and the except block creates the file by opening in write mode, sets the list to an empty list, then closes the file. Without the try and except block the script would stop running if the file did not exist.

  
Figure 1: Try and except block to pen a binary file in read mode

# Raising an Exception for Incorrect Input

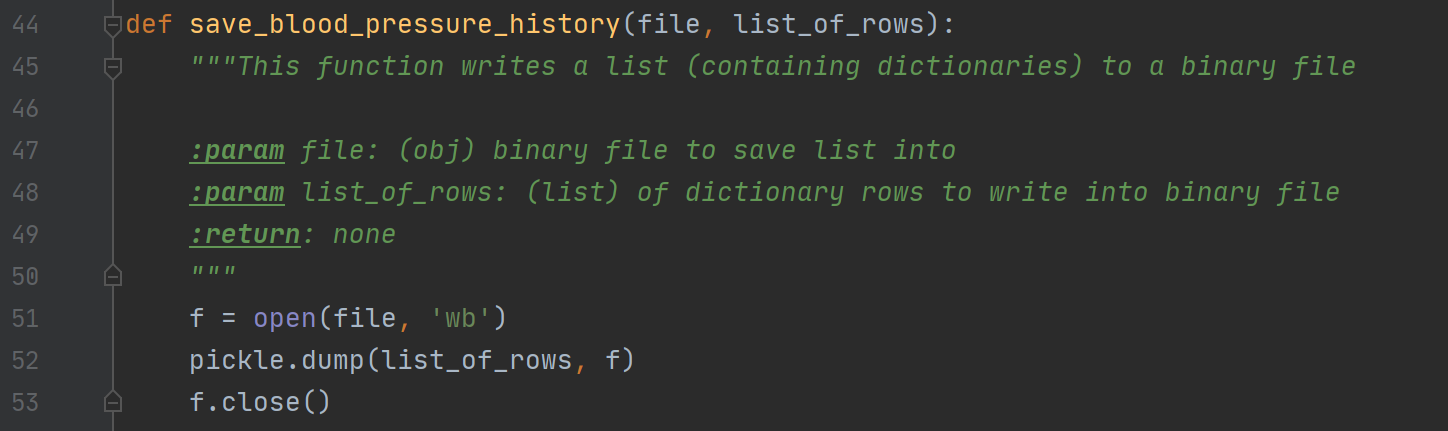
The systolic and diastolic blood pressure readings should only be integer values. Therefore, the blood pressure script created a custom exception where if the user passed a non-numeric value then an exception would be raised and the script would not save the user’s inputted values. Non-numeric covers anything with an alpha character or a number with a decimal point. The custom exceptions are created in rows 69 and 72 of Figure 2. These custom exceptions are called and printed to the user in cases where they input non-integer values.

  
Figure 2: Try and except block to pen a binary file in read mode

If the user inputs two integer values for the blood pressure inputs, then the values are stored to a dictionary and this dictionary is returned.

# Writing to a Binary File

After the blood pressure is entered by the user and added to a dictionary, the dictionary is appended to the existing list of blood pressures stored in the variable blood\_pressure\_history\_list. This list is then written back to the binary file using the pickle.dump(…) command as shown in figure 3.

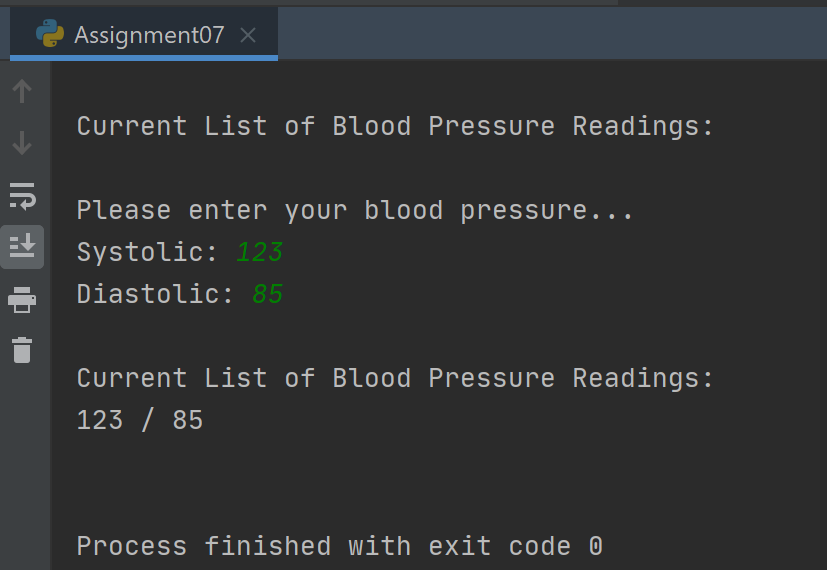
  
Figure 3: Writing to the binary file using pickle.dump(…)

# Running the Blood Pressure Script

The Blood Pressure script can be ran using PyCharm or from a command window. The following provide examples of successful completions of the script using both methods.

## Output using PyCharm

Figure 4 contains a successful run of the script using PyCharm. In this example the blood\_pressure.dat file did not exist in the working directory. Therefore, the script hit the exception and created the file. The initial list of readings was empty, but the list at the end of the run contained the data the user entered as shown in Figure 4:

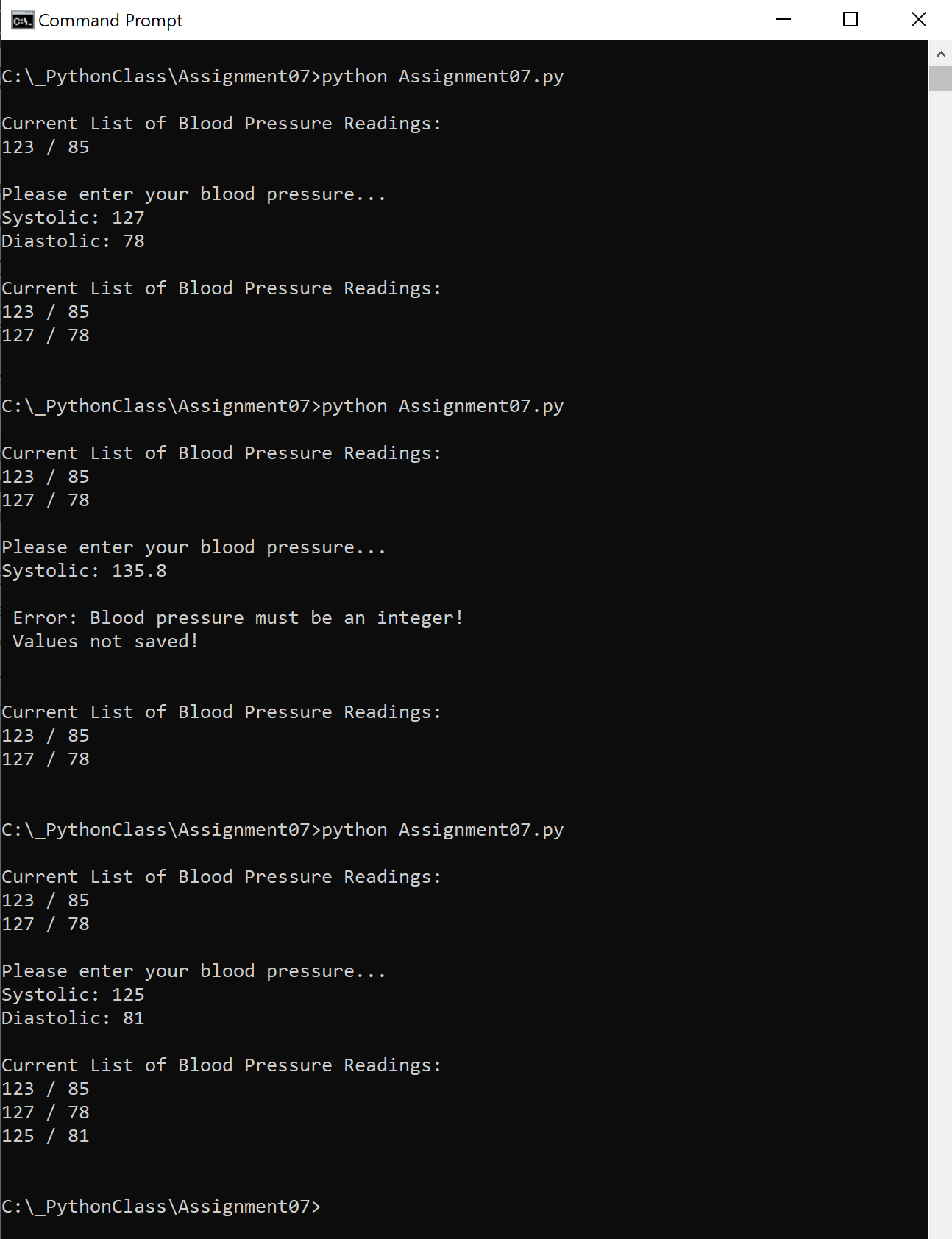
  
Figure 4: A successful run using PyCharm

## **Output using a command prompt**

The following output as displayed in Figure 5 is an example of a successful run using a command prompt to launch the script. When the script launches the blood\_pressure.dat file is found, and the existing data is loaded to a list and printed to the user. The user enters their blood pressure and now two blood pressure readings are displayed back to the user.

The user then launches the script again and the existing two readings are displayed. But upon entering their blood pressure information the user enters a non-integer value and the code catches an exception. The inputs are not saved and the user receives the message: “Error: Blood pressure must be an integer! Values not saved!” as shown in figure 5.

The script is then launched one more time and valid integers are entered. The final result is that the binary file contains a list with three dictionaries stored within.

  
Figure 5: A successful run in a command prompt

# Summary

Using structured error handling is a best practice when writing scripts. The developer of the script should try to think of all the ways that a user could cause an issue when running the code and create exceptions to handle these errors. Pickling is a way to save complex data to binary files. This process requires the importing of the pickling module to load or dump data to or from a binary file.