# **Name:** Karin Sinkevitch

**Date:** August 24, 2020

**Course:** IT FDN 110A

**Assignment:** 07

**GitHub URL:** <https://github.com/KARSINK22/-ITFnd100--Mod07>

# **Binary Files, Pickle Module, and the Exception Class**

# Introduction

This document provides a brief introduction to binary files, the pickle module to read and write binary files, and using the exception class in Python. Reference information can be used to learn more and view examples on these topics. An example script is then described to show how these methods can be applied.

# Binary Files in Python

Binary files store data in a sequence of bytes grouped into eight bits or sometimes sixteen bits. These bits represent custom data and multiple types of data (images, audio, text, etc) can be stored in a single file. Binary files can have custom file formats that must be well understood by any supporting applications. A common example of a binary file is the image file types .PNG or .JPG.

(Reference https://www.thecrazyprogrammer.com/2018/05/difference-between-text-file-and-binary-file.html).

# Binary Versus Text Files

Text files are special subset of binary files that are used to store human readable characters as a rich text document or plain text document. Text files also store data in sequential bytes but the bits in text file represents characters. Text files are of two types: plain text files and rich text files. Both text files store end of line and end of file markers. Figure 1 lists the main differences between binary and text files. (Reference https://www.thecrazyprogrammer.com/2018/05/difference-between-text-file-and-binary-file.html).

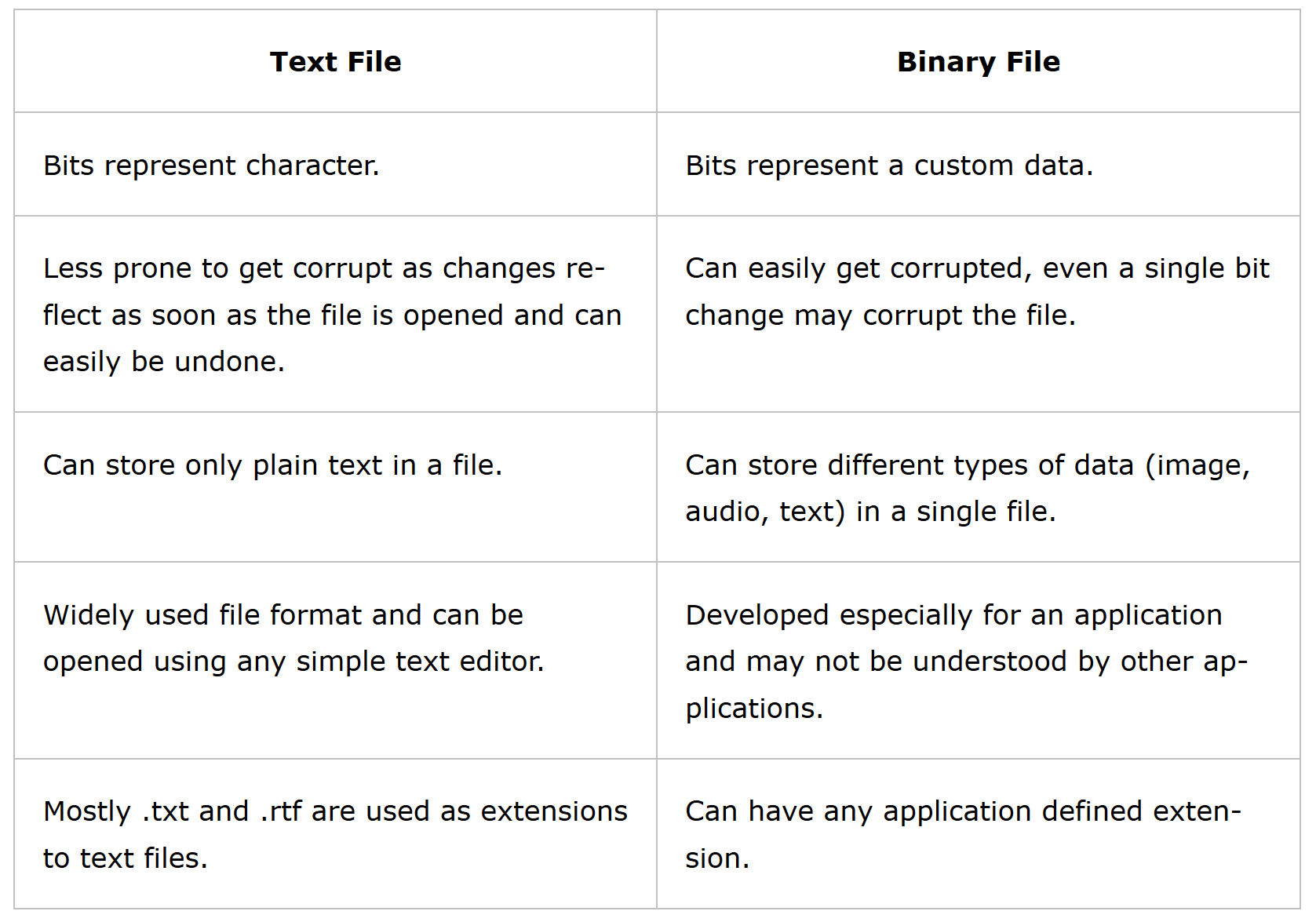


Figure Differences Between Text and Binary File

# Python Pickle Module

Python’s pickle module serializes objects into a binary format so they can be saved to a file and loaded in a program again later on. 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. Pickling is useful for applications where you need some degree of persistency (i.e., the data survives after the process with which it was created has ended) in your data. This data can be saved to disk, sent over a Transmission Control Protocol (TCP) or socket connection, or stored in a database. Using the Pickle module is not recommended if you want to use data across different programming languages. However, do not use the pickle module to deserialize objects from untrusted sources as this can expose you to hacker attacks!

(Reference: [https://www.datacamp.com/community/tutorials/pickle-python-tutorial](https://www.datacamp.com/community/tutorials/pickle-python-tutorial?utm_source=adwords_ppc&utm_campaignid=1565261270&utm_adgroupid=67750485268&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=295208661502&utm_targetid=aud-299261629574:dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=9010513&gclid=Cj0KCQjwp4j6BRCRARIsAGq4yMERCAN_U8zMexDlrsiKmu8J7sjp1yN3QI3GmJl6-n1doNgMFXbre8MaAhaJEALw_wcB) and https://realpython.com/python-pickle-module/).

# Methods

The Python pickle module basically consists of four methods: dump, dumps, load, and loads (Figure 2). The only difference between dump() and dumps() is that the first creates a file containing the serialization result, whereas the second returns a string. The same concept also applies to load() and loads(): The first one reads a file to start the unpickling process, and the second one operates on a string.

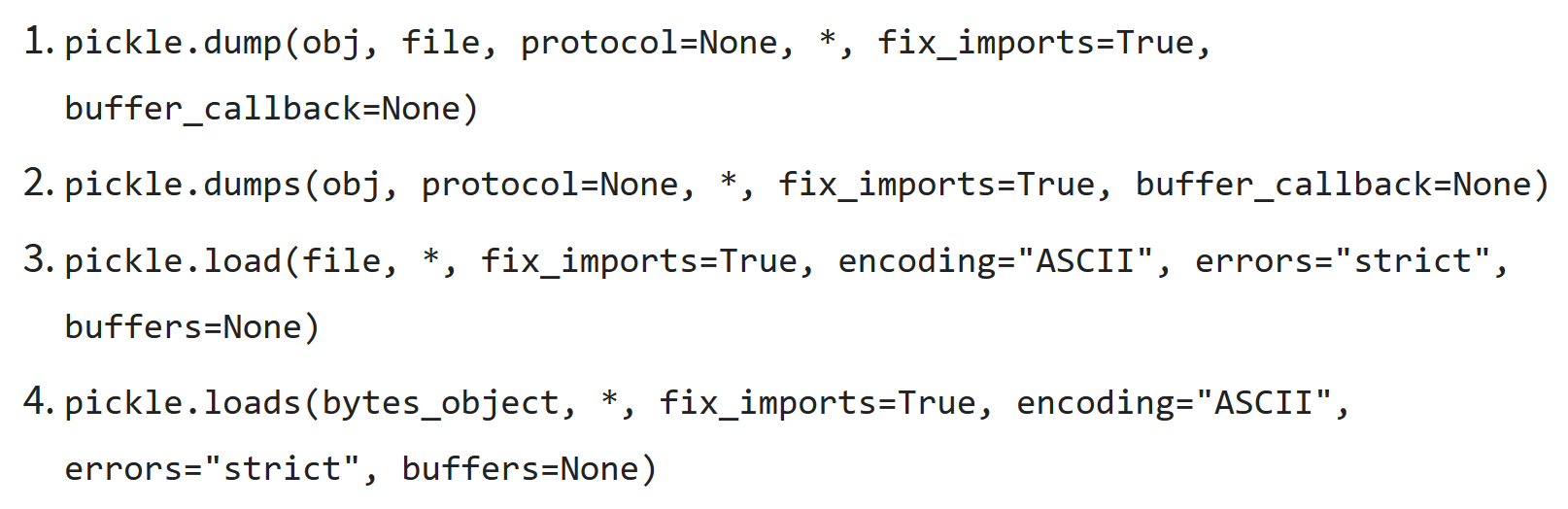


Figure Pickle Module Methods

# Error Handling Using the Exception Class

Python provides two important features to handle any unexpected error in your programs and to add debugging capabilities to them: Exception Handling and Assertions. 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 (Table 1). An exception is a Python object that represents an error. When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible. If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible. If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

You can use a finally: block along with a try: block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. (References: https://www.tutorialspoint.com/python/python\_exceptions.htm).

Table List of Standard Exceptions

|  |  |
| --- | --- |
| **Sr. No.** | **Exception Name and Description** |
| 1 | **Exception:** Base class for all exceptions |
| 2 | **StopIteration:** Raised when the next() method of an iterator does not point to any object. |
| 3 | **SystemExit:** Raised by the sys.exit() function. |
| 4 | **StandardError:** Base class for all built-in exceptions except StopIteration and SystemExit. |
| 5 | **ArithmeticError:** Base class for all errors that occur for numeric calculation. |
| 6 | **OverflowError:** Raised when a calculation exceeds maximum limit for a numeric type. |
| 7 | **FloatingPointError:** Raised when a floating point calculation fails. |
| 8 | **ZeroDivisionError:** Raised when division or modulo by zero takes place for all numeric types. |
| 9 | **AssertionError:** Raised in case of failure of the Assert statement. |
| 10 | **AttributeError:** Raised in case of failure of attribute reference or assignment. |
| 11 | **EOFError:** Raised when there is no input from either the raw\_input() or input() function and the end of file is reached. |
| 12 | **ImportError:** Raised when an import statement fails. |
| 13 | **KeyboardInterrupt:** Raised when the user interrupts program execution, usually by pressing Ctrl+c. |
| 14 | **LookupError:** Base class for all lookup errors. |
| 15 | **IndexError:** Raised when an index is not found in a sequence. |
| 16 | **KeyError:** Raised when the specified key is not found in the dictionary. |
| 17 | **NameError:** Raised when an identifier is not found in the local or global namespace. |
| 18 | **UnboundLocalError:** Raised when trying to access a local variable in a function or method but no value has been assigned to it. |
| 19 | **EnvironmentError:** Base class for all exceptions that occur outside the Python environment. |
| 20 | **IOError:** Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. |
| 21 | **IOError:** Raised for operating system-related errors. |
| 22 | **SyntaxError:** Raised when there is an error in Python syntax. |
| 23 | **IndentationError:** Raised when indentation is not specified properly. |
| 24 | **SystemError:** Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. |
| 25 | **SystemExit:** Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. |
| 26 | **TypeError:** Raised when an operation or function is attempted that is invalid for the specified data type. |
| 27 | **ValueError:** Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. |
| 28 | **RuntimeError:** Raised when a generated error does not fall into any category. |
| 29 | **NotImplementedError:** Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |

# Argument of an Exception

An exception can have an argument, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause. If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception. (Reference: <https://www.w3schools.com/python/python_classes.asp>).

# Deriving a New Class from an Exception Class

Python allows you to create your own exceptions by deriving classes from the standard built-in exceptions. This is useful when you need to display more specific information when an exception is caught. This exception class has to be derived, either directly or indirectly, from the built-in Exception class. When we are developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file. (Reference: <https://www.w3schools.com/python/python_classes.asp> and https://www.programiz.com/python-programming/user-defined-exception).

# Create Python Script “Assigment07.py”

This section describes how to create a script called “Assigment07.py”. The script requests the user to input an ID and customer name. Each set of values is added to a list. Once complete, the user is asked if they would like to save the data to a file. If yes, the data is saved to a binary file. The user is then asked if they would like to have the data read from the file and printed to the screen.

This script provides a great example of using the pickle function to read and write data to a binary file. The functions used in the script are “save\_data\_to\_file” , “read\_data\_from\_file” , and “print\_items\_in\_list”. The presentation section of the script consists of a while loop that requests input from the user. The user is then asked if they would like the data entered saved. Once complete, the user is asked if they would like the data read and printed to the screen (Figure 3).

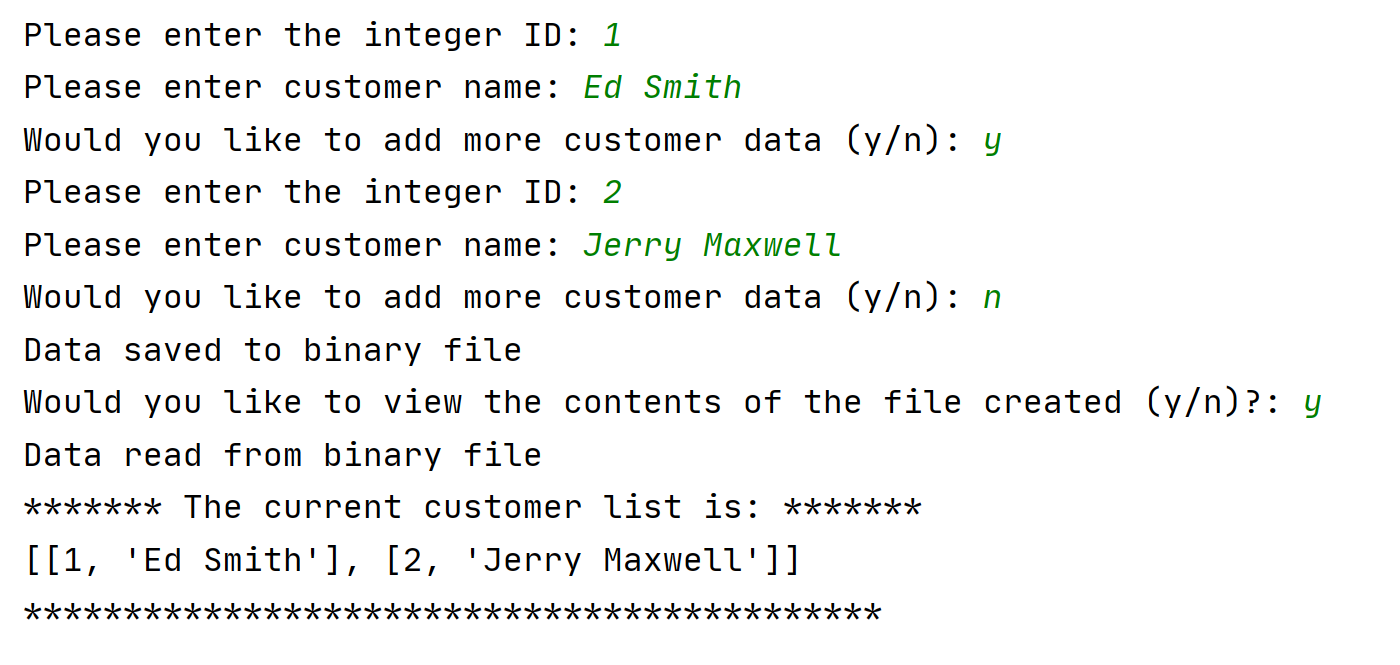


Figure Code Run in PyCharm

Try-except error handling is used when opening, writing, and saving to a file. It is also used to ensure the user inputs and integer value for the customer ID. If a non-integer is input, a value of 1 is given (Figure 4.

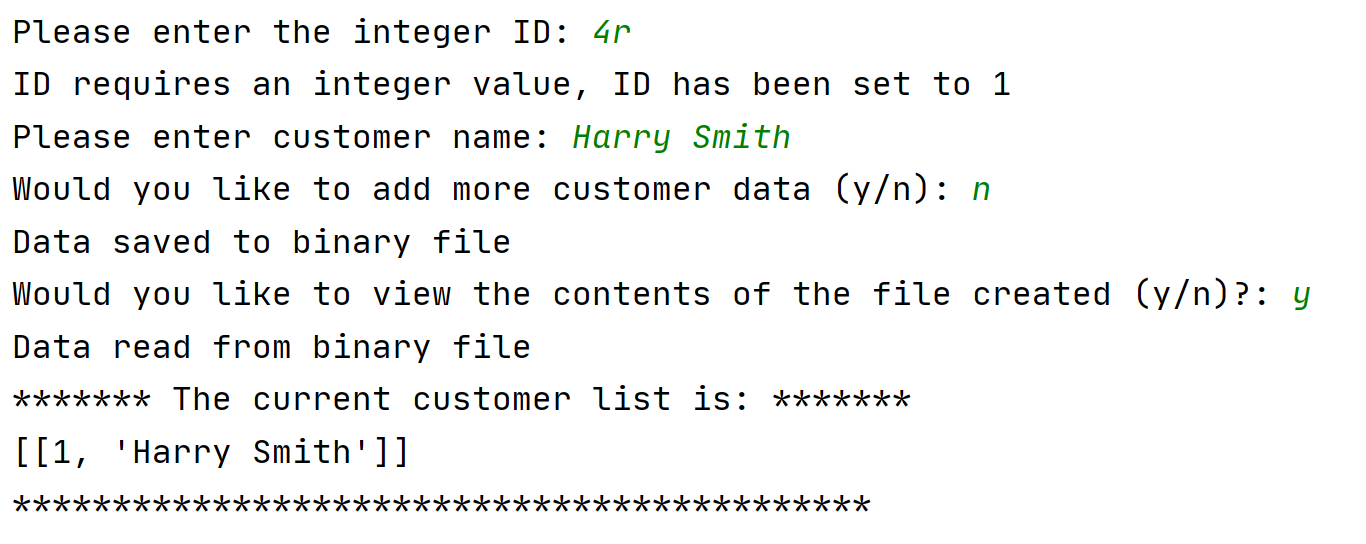


Figure Code with Error Handling of Input

Figure 5 shows the script being run in the console.

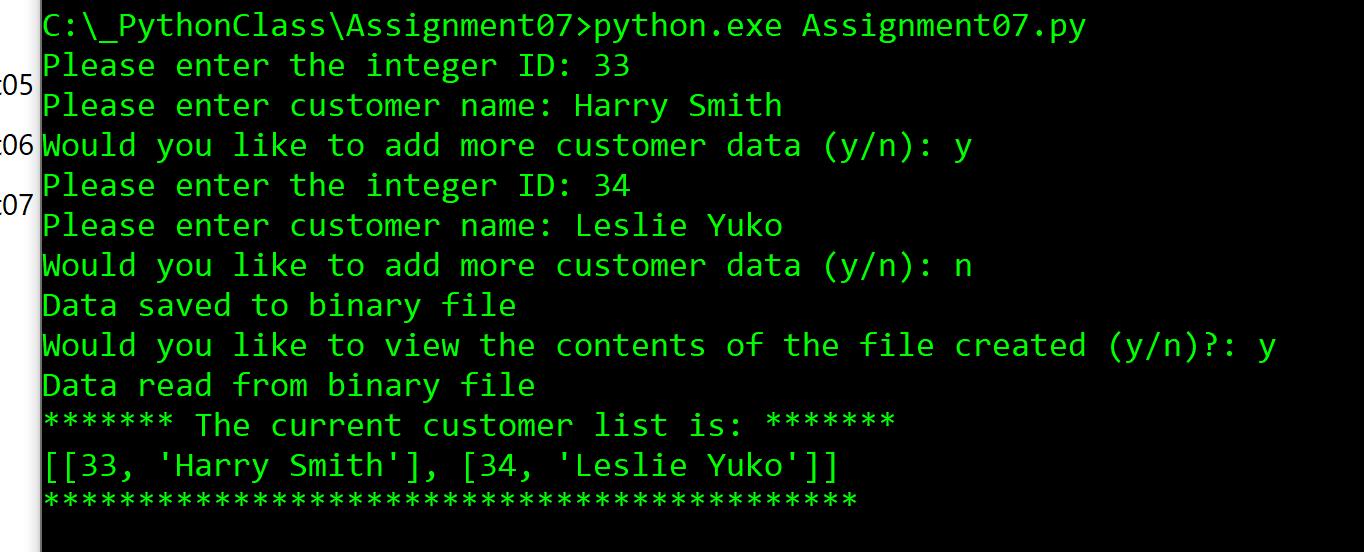


Figure 5 Script Running in Console

# Summary

Reading and writing binary files in Python using the pickle module is very convenient. Exception handling in Python is already robust and can be made even clearer in the right occasions to help the user understand errors when they happen.