Pickling and Error Handling

# Introduction

This document will cover how to pickle data in Python and how to work with error handling in Python. These topics are important for dealing with more than just a text file and building code that is more robust for the user. Pickling allows for use of binary files with Python. Error handling creates a more robust script.

# Pickling

Python has the ability to pickle data. But what does that mean? In the simplest terms it’s a way to preserve data, just as pickling food preserves the food. Pickling is specifically used working with binary data files. When using a binary file, one needs a program of some sort to sort through what is in the binary file and then dump the data back into a binary file format when saving the data.

## Binary vs Text Files

Binary files store the data in the form of bytes (non-text), or 0’s and 1’s. As you can imagine, text files can only contain textual data. Text files however are less likely to become corrupted than binary files due to the structure and detail that binary files must be written in. A small error in a text file may cause a slight error, but a small error in a binary file can cause the file to not be usable.

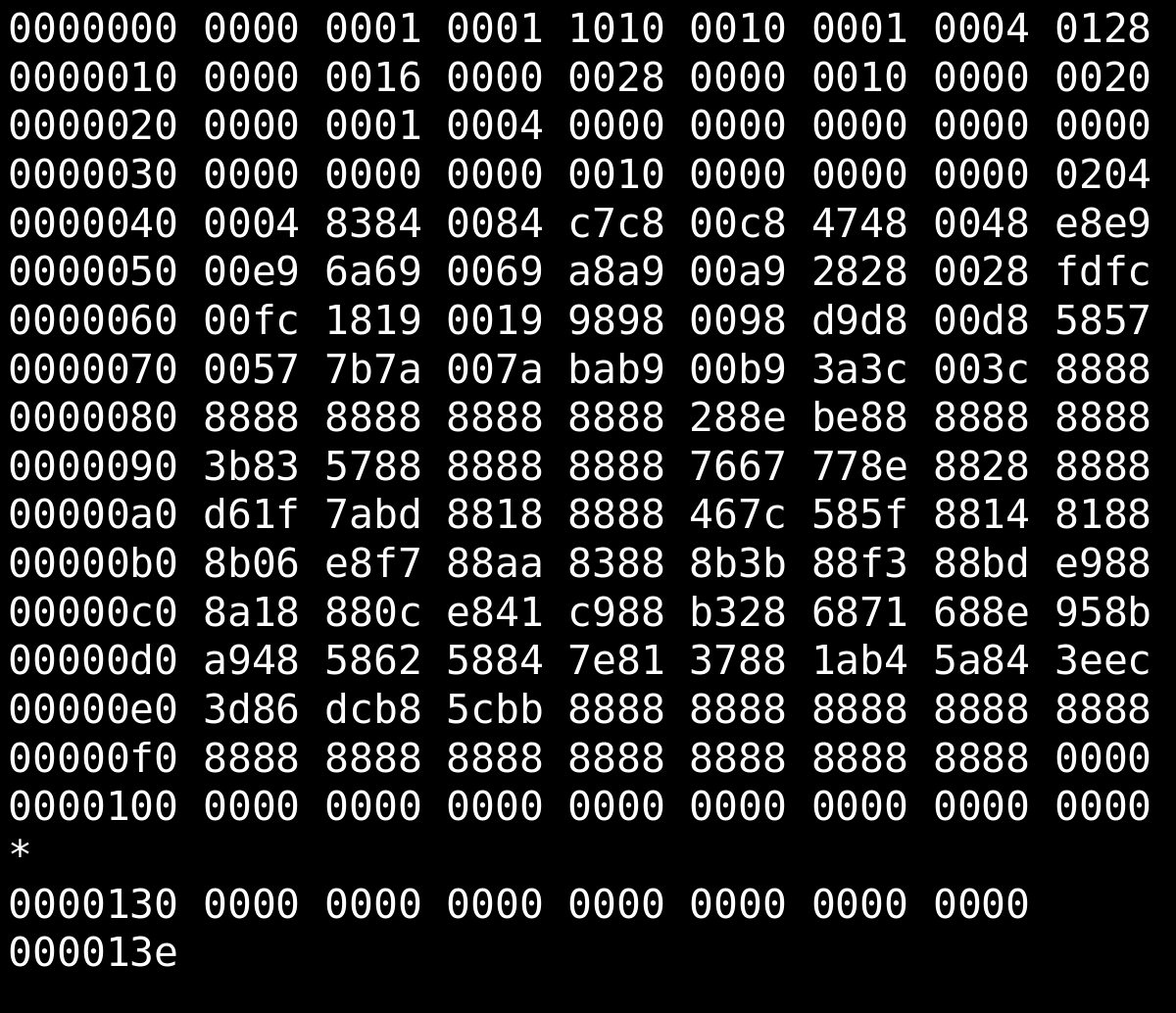


Figure 1. Example of what a binary file looks like. This example shows how much more difficult it is to read a binary file than a text file. (Source: [Wikipedia](https://en.wikipedia.org/wiki/Binary_file))

Binary files are useful because they can be saved to a disk or easily sent to others. Binary files should not be used when trying to share code across different coding languages. The Pickle protocol is only used in Python, so the code will not work across different languages.

For more information on binary files including, what they are, advantages over text files, and issues with binary files; I read over a CareerKarma article ([CareerKarma article](https://careerkarma.com/blog/what-is-binary-file/)). This site also had some great information about text files as well. I liked that the article went over both types of files and gave pros and cons of each file type.

## How to Pickle Files

To start pickling a file, one must start by importing it into Python. This can be done with a simple line of code that can be seen in the below figure.



Figure 2. This is an example of the script needed to be able to start pickling data.

The script needed to read, write, and append data to or from a binary file is very similar to how to read, write, and append the data to a text file. The main difference is the addition of a “b”. See table and figure below for examples of the script. A more detailed table can be found on page 201 in Python Programming for the Absolute Beginner, 3rd Edition.



Figure 3. Example script of how to open a binary file in read mode.

|  |  |  |
| --- | --- | --- |
| **Mode** | **Binary** | **Text** |
| **Read** | “rb” | “r” |
| **Write** | “wb” | “w” |
| **Append** | “ab” | “a” |

Table 1. Example of the differences between reading, writing, and appending a binary or text file.

Binary codes are complicated, so I’m first going to go over how to use Python to create a binary file. To start with, we will need to import “pickle” to be able to use it (see Figure 2) and then ask Python to write a variable, list, dictionary, or value to the binary file. The convention Python uses is dump - dump the data from the “list\_of\_rows” into the file.

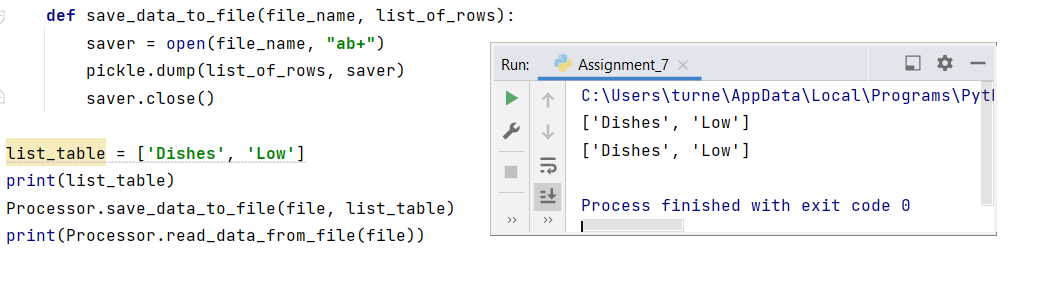


Figure 4. Example of the script for the function to save the data to the binary file and the function to read the data from the file. Both a simple print function and the read from file function both produce the same result.

With data in the file, I now wanted to read the data back out of the file to check that the save was successful. In Figure 4, I read from the file and printed the information back. As with saving data to the file, reading data from the file is very similar to the script used for a text file. Open the file, place the data in the file into “list\_of\_rows”, and then return “list\_of\_rows”.

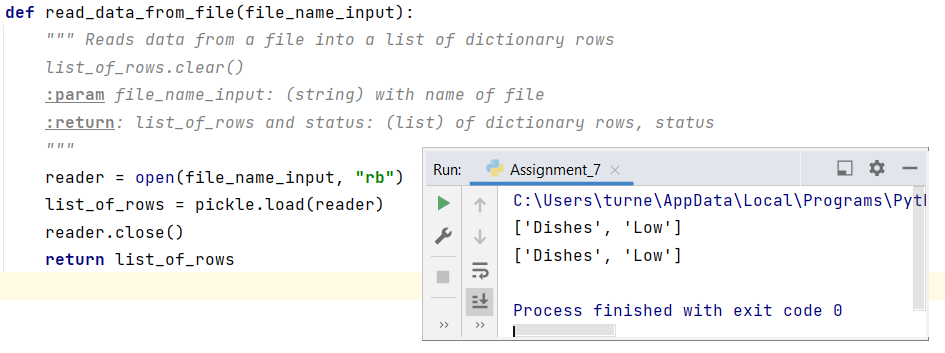


Figure 5. Example of the script used to read the data from the binary file into the “list\_of\_rows”. The figure also shows the output from the simple print of the list and the print from the reading of the list from the binary file.

As demonstrated in this section, reading/writing/appending to a file (binary or text), is similar with some differences. First difference is you must import pickle at the start of your code. Pickle is always needed when working with binary code. Next there is a difference in how you open the file, there is an added “b” to the letting, as seen in Table 1. The file extension for a binary file is “dat” instead of “txt” for a text file.

# Structured Error Handling

Errors occur when running scripts and programs. Errors are normal. In Python when it comes to an error, it will throw an exception. An exception meaning it found an issue and doesn’t know how to resolve the issue. When Python finds the exception, the program or scripts exits, and Python prints out a non-user-friendly exception error.

Graphical user interface, text, application, email

Description automatically generated

Figure 6. This is an example of the exception when a file is not found.

In Figure 5, Python was asked to read the file Error.txt, but since the file doesn’t exist yet, the file can’t be read. Python throws an exception and prints out some information about the exception. The hard part about this exception is understanding what it means.

The final line in the output in Figure 6, states that the program ended. With the program ended, the user cannot continue on with the program. To deal with complicated exception statements and to keep the program running, there are two things as coders we can do to improve the functionality of the script.

First, we can ask Python to print out more information than the standard output from an exception. The second thing we can do is use a try/except block to allow the program to keep running even if there is an exception. The final step a coder can do is create custom exceptions; this is especially helpful when requiring specific types of inputs from a user such as a numeric value.

## Try/Except Blocks

To keep the program running for the user, try/except blocks are used to let the code try the script and if it doesn’t work do something else. The example error we started with in Figure 6, was file not found error. If the program was used to read information from a file, change the data, and then save it back to the file; if there was no file to read the program would just end and the user wouldn’t be able to complete any other tasks.

To fix this probably of the exception we can use a try/except block to try and read the file, and if there is an exception print a message back to the user and continue through the code. An example of this script is in Figure 7 below.

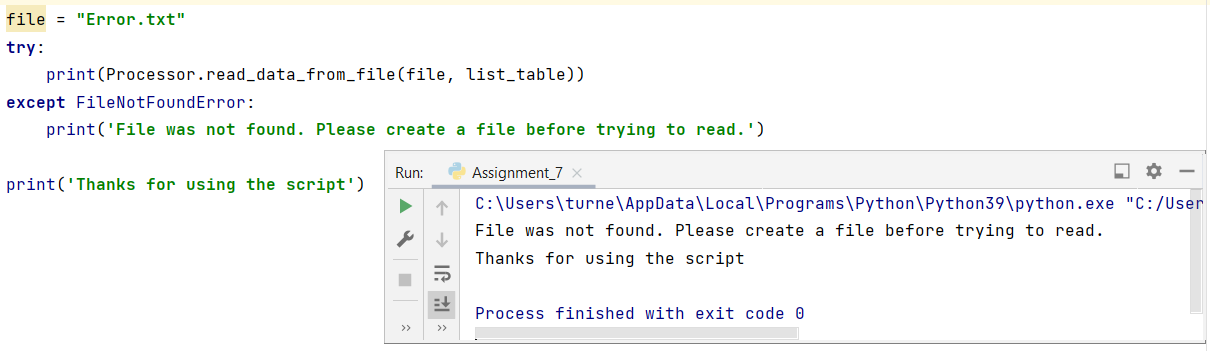


Figure 7. This is an example of the script and output from the try/except block.

The script asks for the file “Error.txt” to be read, but the file does not exist yet, so Python can’t read it. The except block catches that and returns a more friendly note to the user. In this example we already knew what the exception was based on previous trials to read the data. What happens when we need more information to figure out what Python is trying to tell us.

## Information about Exceptions

One of the most helpful things one can do when wanting to know more information is to print it out. A try/except block can be set-up to print information out if there is an exception. Based on the additional information we can be more specific with the except script.

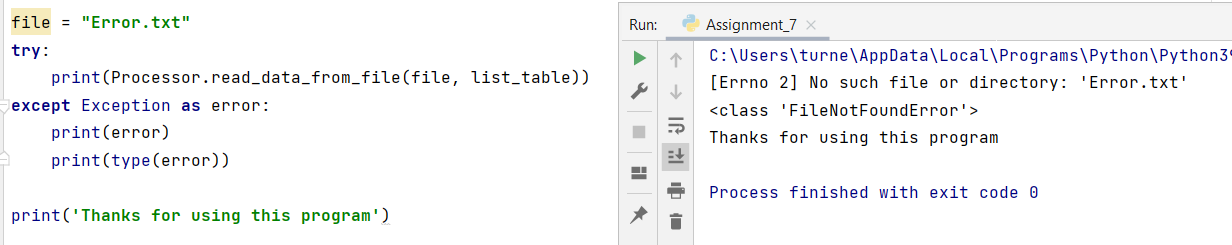


Figure 8. Example of how to print information about the error, to help understand the exception and how to deal with it. The exception is “No such file or directory: ‘Error.txt’” and the type is in the class <FileNotFoundError>.

With information about the exception printed, it makes it easier to figure out what can be done about the exception. In the example in Figure 8, the error type printed was “FileNotFoundError”. Now that I know what the exception is a I can write a more specific except script to provide input back to the user. An example of that is shown in the previous Figure 7.

## Customizing Exceptions

Python also allows custom exceptions to be raise. If one wanted the user input for a certain variable to only be a float and nothing else, an exception could be called if the user entered something other than a number.

Graphical user interface, application

Description automatically generated

Figure 9. Example of how to write the script to raise a custom exception based on if there is an issue with the input or variable.

There are several ways one can create custom errors. Above is a simple version of error handling, there are several other ways to write custom errors. Programwiz has a great article on how to step through the different methods for error handling (External Source: [Programwiz Link](https://www.programiz.com/python-programming/user-defined-exception)). The Programwiz article also provides a lot of helpful information about what the script examples are trying to accomplish.

As we did in Figure 9, we are returning a simplified message back to the user. What wasn’t covered in Figure 9 example, was declaring the class for the exception. Python has many built in exception classes that can be used. An example of this is given in Figure 7. When a file is not found, it can’t be read, Python’s built-in error handling exception is “FileNotFoundError”.

To show how both built-in and custom error can work, I created an example. The example is a program to make a shopping list. I’m going to ask the user to an item to add to the shopping list and the quantity of the item. Now it’s time to figure out what error handling might be useful. I want the user to enter quantity as an integer. From the Python website, there is a list of built in exceptions and how they work (External Source: [Python Exceptions](https://docs.python.org/3/library/exceptions.html#exception-hierarchy)). One the list there is a built-in error for ValueError. I can also write the code to ask for an integer input, and see what error message is given back, as seen in Figure 8.

The next error handling step I want to take, is to not let the user enter anything less than 1. When shopping we don’t normally return items once on the shopping list, and if items are on the list, we normally want at least 1 or more. To make sure the user enters an input for quantity larger than 1, I can create a custom error class and print my own output when that exception is raised in the try block.

In the below figure, the CustomError class is defined, and a custom message is printed. The second part of figure below, shows the try block. In the try block, the user enters a quantity that I have defined as an integer. If the input is an integer, the program then tests to make sure it’s less than 1. If it is less than 1, my CustomError is raised. If the user doesn’t enter an integer, the try block fails, and except script for ValueError runs and prints a message letting the user know not integer was entered.

Graphical user interface, text, application, email

Description automatically generated

Figure 10. Examples of built-in (ValueError) and custom error handling script.

The drawback from using the script in Figure 10 is that the script stops when it gets to the error. When a user is running the program, it’s inconvenient for the user to keep starting the file over when there is an issue. To help the program keep running a while loop can be used to make sure the user gets to try again till the input is no longer an issue.

Graphical user interface, text, application

Description automatically generated

Figure 11. Using a loop, try/except block, built-in error handling, and custom error handling. This script asks the users for a quantity, checks it’s an integer and that’s between 1-10.

## Testing and Finding Potential Errors

The last part of custom error handling is to figure out what custom errors you need. One way to do this is to test your code and run in through all the options to see where the weakness lies. Also asking a friend to test your code is never a bad idea. Since they didn’t write it, sometimes it is easier for them to find the bugs.

The example I’m going to use is a shopping list. From previous information and assignments, we know how to make a list of tasks with priorities. A shopping list isn’t too different from that, just updating some of the code for it to make sense. I want to use pickling to read and write my list to a binary file. Since we want to change the list, I’m going to make an option to add and remove items.

The first part of my script reads data from a file. I can think of a few ideas of how this script could be broken. If the file isn’t a binary file, then I don’t want to save data to it for whatever reason. Second if the file doesn’t exist yet since I haven’t figured out what I need, then another error will be thrown since Python can’t read it.

For the file name, I could create a custom error for how the file ends and if it isn’t .dat loop the user until they enter a correct type. Reading a file that doesn’t exist is a built-in error handling exception called FileNotFoundError, as seen in Figure 7 and Figure 8.

The next spot for issues comes with user inputs. The user can enter whatever they want to, we have no control. To help the program be more robust, we can check the user input based on defined limits/controls, as seen in the last section. I could also look at limiting the inputs for the menu options, but for right now my loop catches that potential issue.

As we ask the user for more input and create more complex scripts, the need for error handling grows. Without good error handling, the script and program won’t be as robust as it could be.

# Summary

In this document we covered how to pickle a file and how to work with error handling. Pickling allows for data to be read, written, and appended to binary file or a text file. In order to help improve robustness of programs, error handling is important. Error handling lets us deal with bad inputs to the user or prevent the program from ending early due to other issues such as files not existing. Python has several built-it error handling exceptions, but we also learn how to create custom error exceptions. All the information reviewed in this document will help limit the number of issues a user might run into.

THANKS FOR READING!