

Pickling and Error Handling

Introduction

This paper provides a basic overview of Python's pickle module and how it can be used to save and access binary data, as well as some examples of error handling and Python's "try/except" structure. While this particular script utilizes a number of other Python features, this paper will mainly focus on describing the sections that involve pickling and structured error handling.

Pickling

To understand pickling in Python, I supplemented the provided information from the textbook and lecture materials with some of my own research. A couple pages I found useful, due to their simplicity and clarity, were <https://www.datacamp.com/community/tutorials/pickle-python-tutorial/> (external link) and <https://www.tutorialspoint.com/python-pickling> (external link). These pages provide an overview of what pickling can be used for (and when it should not be used) and provide several examples of the pickle module in action.

The first step for using Python's pickle module is to import it into the script. I placed this import at the start of my script near where I declared my variables (Figure 1).



```
# --- Data --- #
import pickle

string_file_name = ''
string_title = ''
reading_lst = []
```

Figure 1: Importing the pickle module

To perform my pickling tasks for this script, I placed the relevant code in functions within a processing class. To pickle data to a file, my function requires the file name and the data to be pickled as its parameters. In this case, the data is a list, and the file name is retrieved from a separate function in the presentation class capturing user input.

I open the file using the same general format as writing to a text file, but instead of using 'w', I use 'wb', in which the 'b' stands for binary. This tells the program that the data will be written in binary, or byte streams. Next, I use the dump function to write the data to the file before closing it (Figure 2).

```

@staticmethod
def pickle_to_file(file_name, list_of_titles):
    file = open(file_name, 'wb') # open the file for writing
    pickle.dump(list_of_titles, file) # "dump" pickled data to file
    file.close() # close file
    print('\n' + 'Reading list saved to file.')

```

Figure 2: Defining a function to pickle data to a file

To retrieve the binary data from a file, I use a function requiring the name of the file as its parameter. I open the file and this time use the notation 'rb' for reading from binary. Then, the pickle load function allows the program to unpickle the file data and load it into the program. As always, I close the file after the data is retrieved (Figure 3).

```

@staticmethod
def unpickle_from_file(file_name):
    try:
        file = open(file_name, 'rb') # open the file for reading
        file_data = pickle.load(file)
        file.close() # close file
        print('\n' + 'Reading list loaded from file.')
        return file_data # return unpickled list
    except FileNotFoundError:
        print('\n' + 'File not found: Please choose an existing file.')
    except pickle.UnpicklingError:
        print('\n' + 'Data error: Please check file for corruption.')

```

Figure 3: Defining a function to unpickle data from a file

I call these functions for pickling and unpickling in the main body of my script (Figure 4).

```

elif menu_choice == 4:
    IO.input_file_name()
    Processor.pickle_to_file(string_file_name, reading_lst)
    continue

elif menu_choice == 5:
    IO.input_file_name()
    Processor.unpickle_from_file(string_file_name)
    continue

```

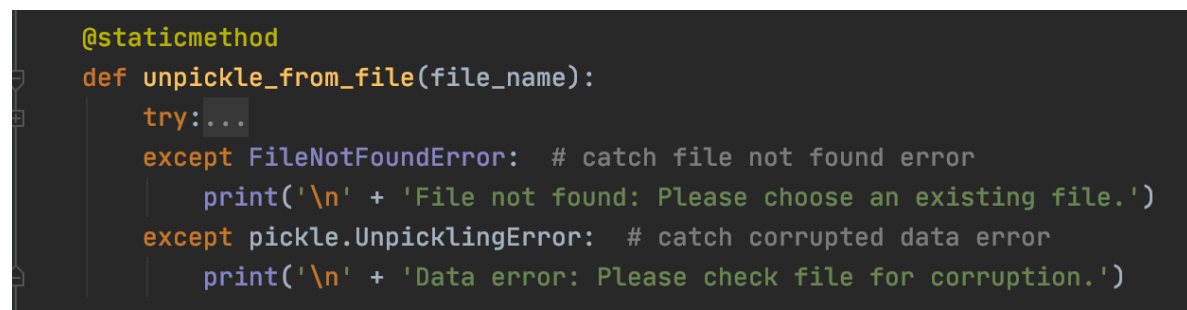
Figure 4: Calling functions to pickle or unpickle data from a file

Structured Error Handling

This script contains several “try/except” blocks, a form of structured error handling in Python. These sequences are intended to anticipate particular errors that may arise in the program and “catch” them so that they do not cause the program to fail and terminate. A handy feature is being able to create custom error messages for these exceptions so that they display more user-friendly information, rather than (or in addition to) Python’s built-in error messages.

A few resources that I found helpful for learning about error handling were <https://www.datacamp.com/community/tutorials/exception-handling-python> (external link) and https://www.w3schools.com/python/python_try_except.asp (external link). These pages explain the use of raising and handling exceptions for a variety of common errors.

In my function for unpickling data from a file, I utilized a try/except block to anticipate the user entering a file name that cannot be reached by the program (Figure 5). This same block includes a second except statement for situations in which the file data is somehow corrupted and cannot be unpickled.



```
@staticmethod
def unpickle_from_file(file_name):
    try:...
    except FileNotFoundError: # catch file not found error
        print('\n' + 'File not found: Please choose an existing file.')
    except pickle.UnpicklingError: # catch corrupted data error
        print('\n' + 'Data error: Please check file for corruption.')
```

Figure 5: try/except block for catching errors while unpickling data from a file

In the main body of my script, I use a try/except block to anticipate the user entering something other than the numbers 1-6, which correspond to the available menu options (Figure 6).

```

try:

    if menu_choice == 1:...

    elif menu_choice == 2:...

    elif menu_choice == 3:...

    elif menu_choice == 4:...

    elif menu_choice == 5:...

    elif menu_choice == 6:...

    elif menu_choice != 1-6:
        raise IndexError()

except IndexError as e:
    print('\n' + 'Index error: Please enter a number from 1 to 6.')

```

Figure 6: try/except block for catching user input errors

Similarly, because I converted my user input for menu choice to type *int*, I included an exception to catch input errors that could not be converted to type *int* (Figure 7).

```

@staticmethod
def input_menu_choice():
    try:
        choice = int(input('Which option would you like to perform? [1 to 6]: '))
        return choice
    except ValueError:
        print('\n' + 'Value error: Please enter a number.')

```

Figure 7: try/except block for catching user input errors

Running the Program

Once I defined all my functions, I ran all aspects of the program in PyCharm. When the user selects option 5, they are prompted to enter a file name, and data is unpickled from that file and loaded into the program (Figure 8).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 5

Enter file name: books.dat

Reading list loaded from file.
```

Figure 8: Option 5 running in PyCharm

If the user enters a file that does not exist, however, or is unreachable by the program, an exception is raised and the user is notified (Figure 9).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 5

Enter file name: movies.dat

File not found: Please choose an existing file.
```

Figure 9: Option 5 running in PyCharm with exception raised for `FileNotFoundError`

When the user selects option 4, they are prompted to enter a file name, and data is pickled to a file in binary (Figure 10). In this case, the user can enter either an existing file name, which will overwrite any existing data in the file, or a new file name, and the program will create that file in a relative path.

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 4

Enter file name: books.dat

Reading list saved to file.
```

Figure 10: Option 4 running in PyCharm

When the user selects option 2, they are prompted to enter a title, which is then added to the list (Figure 11).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 2

Title: Girl, Woman, Other

Title added to reading list.
```

Figure 11: Option 2 running in PyCharm

When the user selects option 3, they are prompted to enter a title, which is then removed from the list (Figure 12).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 3

Title: Girl, Woman, Other

Title removed from reading list.
```

Figure 12: Option 3 running in PyCharm

When the user selects option 1, the current list is displayed (Figure 13).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 1

Reading List:
* Normal People
* My Own Words
* The Testaments
```

Figure 13: Option 1 running in PyCharm – displaying current list

If there are no items in the current list, the user will be notified (Figure 14).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 1

List is currently empty.
```

Figure 14: Option 1 running in PyCharm – empty list

When the user selects option 6, the program ends and the user is notified (Figure 15).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 6

Goodbye! Happy reading!
```

Figure 15: Option 6 running in PyCharm

After running the program in PyCharm, I also ran it in Terminal, in the corresponding directory.

The user can start with an empty list, then load data from an existing file to populate the list (Figure 16).


```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 1

List is currently empty.

Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 5

Enter file name: books.dat

Reading list loaded from file.

Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 1

Reading List:
* Normal People
* My Own Words
* The Testaments
```

Figure 16: Option 5 running in Terminal

The user can add and remove titles from the list (Figure 17).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 2

Title: 1493

Title added to reading list.

Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 1

Reading List:
* Normal People
* My Own Words
* The Testaments
* 1493

Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 3

Title: 1493

Title removed from reading list.

Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 1

Reading List:
* Normal People
* My Own Words
* The Testaments
```

Figure 17: Options 2 and 3 running in PyCharm

And the user can save created lists to a file (Figure 18).

```
Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 4

Enter file name: books.dat

Reading list saved to file.

Options
1 - Display reading list
2 - Add title to list
3 - Remove title from list
4 - Save list to file
5 - Load list from file
6 - Exit

Which option would you like to perform? [1 to 6]: 6

Goodbye! Happy reading!
megan.m.giddings@Megans-MacBook-Air Assignment07 %
```

Figure 18: Options 4 and 6 running in PyCharm

To verify that the list was saved, I checked the specified file for the binary data (Figure 19).

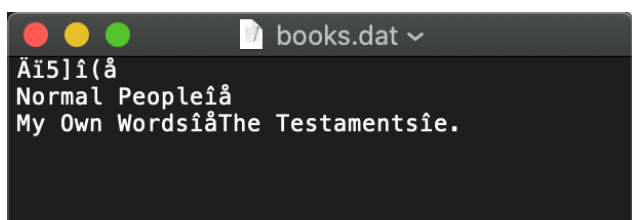


Figure 19: Verifying byte stream (binary) data saved to file

Summary

This paper has provided an overview of the use of pickling binary data and some examples of structured error handling. While pickling works similarly to writing to a text file, there are some different nuances, and it can be important to understand when to use which form when saving and accessing data. Structured error handling is an extremely useful technique for writing more efficient programs, and it can help the developer better understand the potential cracks in their code.