Kevin Scales

Mar 2, 2021

Introduction to Programming (Python)

Assignment 7

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

Pickling, Exceptions, and the Worst Encryption Ever

# Introduction

The main topics for this assignment are pickling, a method to save objects in Python in a binary format, and exceptions, ways to handle the appearance of an error during run time other than crashing the program. It does so with a fanciful application to encrypt text messages by pickling them and saving in a presumably unreadable to human eyes binary format. Since the encryption is not serious but rather a fun way to explore the programming topics, we can state upfront that pickled text is not always very hard to read at all, as the examples will show.

# The Python Research

The first part of the assignment was to do some online research and find sources of pickling and exception handling. I feel compelled to say that I didn’t come across any bad ones during the time I spent. A decent-enough source was the group python.org, which unsurprisingly holds a vast supply of information. In <https://docs.python.org/3/tutorial/errors.html>, I found clear cut examples of try-except code and hints like using raise to explore specific error code handling options. A corresponding page at <https://docs.python.org/3/library/pickle.html> had greater detail about pickling but, as might be expected from the name, less readability than the corresponding tutorial page had about exceptions.

For easy daily access, I found not a website but rather the (very large) book [Learning Python, by Mark Lutz](https://www.amazon.com/Learning-Python-5th-Mark-Lutz/dp/1449355730). The main advantage is the combination of thoroughness and readability, with the main drawback being one has to purchase it first and have it in the room.

# The Python Code

The open nature of this assignment inspired me to have a little fun and imagine a program that ‘encrypts’ string data by pickling and saving it as binary. Anyone who has ever opened a binary file thinking it a text file knows that what results can be essentially unreadable. Although this proved not quite to be the case with binary pickled files, it demonstrated file management and error handling in an amusing manner.

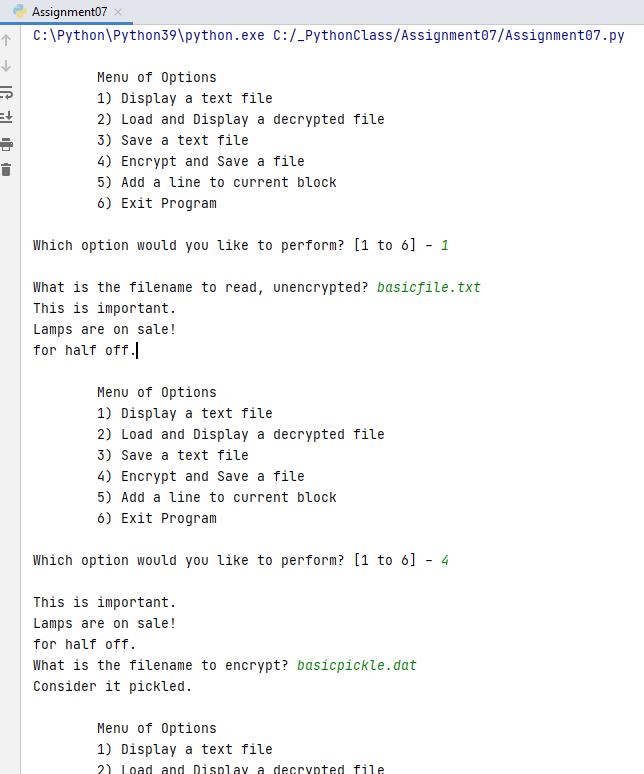
The program starts with (and always comes back to) a menu with six options, two open a text or ‘encrypted’ file, to save a text or ‘encrypted’ file, and to add new information to whatever is in memory now. The last option is to exit (an easy feature needing no new description).

Some of the overall structure is borrowed from the last assignment. A class called Processor contains functions to read and write files and add data, while IO shows the menu and gathers information. The first defined function, to read text data, also includes the first example of exception handling. A try block opens and reads a file, hopefully, returning the data and the message ‘Success’. But if there is no such file as requested, it returns the unfilled variable and ‘Fail’. If some other error occurs, it returns ‘BadFail’. The first menu choice in the main section calls this function and responds not only to the returned data, which it prints out if there’s anything to print. But if the returned second item, called flag here, indicates a failure, it gives a message about no file existing and then moves on. Alternatively, it gives a stronger warning message about an unidentified error and a warning about continuing onward. I used similar try-except blocks in other parts of the code. In theory, professional code should have error handling options built into virtually every portion of the code, but this requires constructing one time a lot of functions that can then be called repeatedly. For example, writing in C years back, we never wrote code to set x equal to y/z. We called a small function and passed y and z, with a proper response if z was zero. We did not even printf anything. We called a KernelMessage function (or something like that) with the text to display, in case printing failed. It was very thorough.

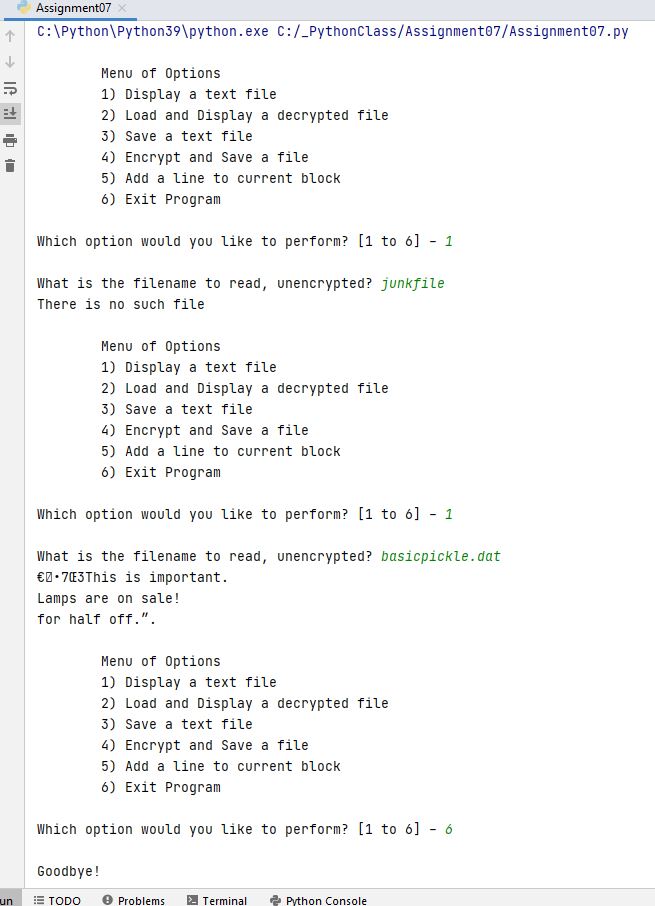
For pickling, the ‘encrypted’ text is just pickled string data written to a binary file, or read from one. Because pickling is its own function, the calls and reads are very short, to the point calling a function would be extraneous, so these were handled mainly in the main body. The methods pickle.load and pickle.dump quickly and efficiently did all the work. One feature of handling files and data as I did was that it did not check the format of the files. This allowed us to read the raw ‘encrypted’ data as if it were a text file. It also demonstrated that even binary files have sections of readable code when read into a Python program, or into Notebook, as the screenshots will demonstrate. When the entire content is just short text lines, it is not hard to decipher them.

# Results

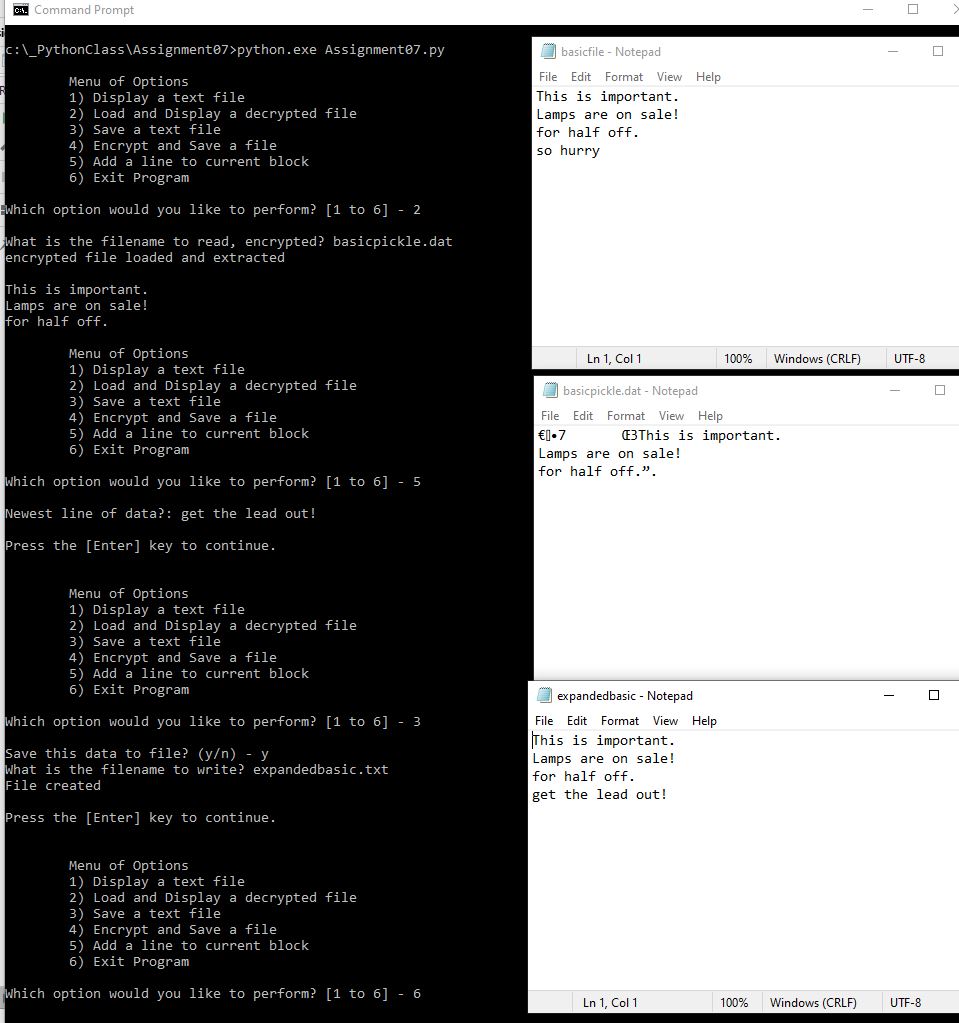
In the first shot, I open an existing text file and read the contents. Then I save them via pickling in an ‘encrypted’ style.



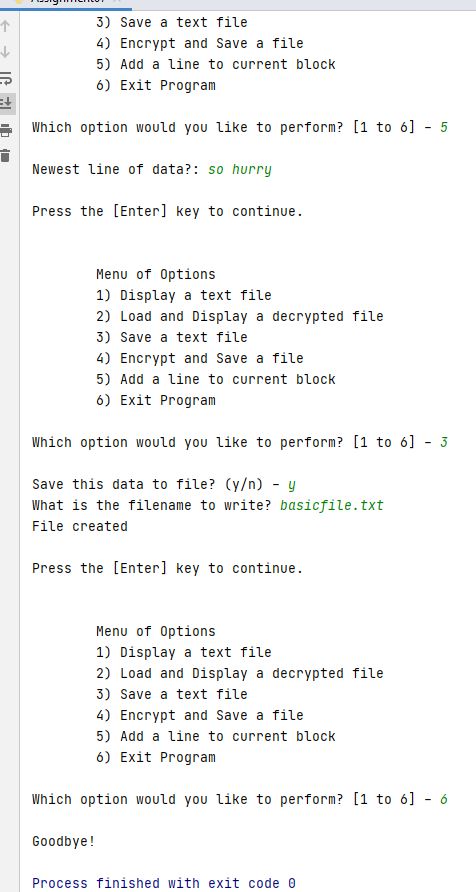
In the second shot, I try to open a file that does not exist. The exception handling takes care of it and allows me to continue by reading in an ‘encrypted’ file as a text file. The results show that it is binary, but it doesn’t hide much.



Switching to a command prompt, I load the binary as a binary and display the data, much cleaner. I add some information and save it as a new text file. The actual files in Notebook verify the contents. (For the first one, basicfile.txt, it shows the results from the next block below)



Finally, back in PyCharm, I add some other text and save as a text. The Notebook in the previous figure shows this outcome.



# Conclusion

Pickling allows us to write complicated data structures to files in a compact way that strings don’t really show us, but nonetheless exists. It is also a very simple way to read and write data even for simple data types if the saved filed does not need to be read directly by people. It would not be suitable for creating a ReadMe file, for example, but can quickly and easily save and reload large blocks of data.

Exception handling clearly is a vitally important task, and I introduced it in this assignment in a few locations where it could clearly showcase its importance, like continuing the program even after trying to open a non-existent file. Future work, in this class and in applications outside of class, can and should continue to build up a set of error handling protocols, perhaps in my own personal repository, that can be used over and over. The Python environment makes it easy to see what kind of errors can arise and how, making trapping for them all the easier.