Bramble Eagle-Jack

2021-02-22

Foundations Of Programming - Python

Assignment 7 Knowledge Document

Files and Error Handling

# Introduction

This week we’re asked to review learning materials that give us an in-depth look at files and error handling. The assignment this week asks us to use the newly learned concepts to modify an existing script so it includes a binary data file and error handling.

# Reading and Writing to File

Text files are opened with the open function which uses two arguments to return a file object. It’s good practice to use the close method on all files that are opened unless opened with a with statement. The first parameter is the file name string to be opened and then a string indicating the mode in which the file should be opened. The string ‘r’ is the mode for reading a file and the string ‘w’ is the mode for writing to a file. Both modes allow me use to several mode-specific methods with the file object.

In read mode, I can read the whole file with the read() method and return its contents as a string. Readline() is another method that reads a single line every time it is called. The cursor moves toward the end of a document after reading so the readline() method will read the next line each time it is called. The final method for reading a file is the readlines() method which reads the entire file and returns each line as an element in a list. Using a number as an argument in either the read() or readline() method returns that many characters from the file or line being read. The same concept of the cursor moving after reading applies to characters as well. I can also use a for loop without any of the read methods and iterate through the file lines as elements and assign them to a list.

1. num\_str = '1234\n'
2. str\_list = ['abcd\n','efgh']
3. e\_lines = ''
5. with open('file.txt', 'w') as file\_obj:
6. file\_obj.write(num\_str)
7. file\_obj.writelines(str\_list)
9. with open('file.txt', 'r') as file\_obj:
10. a\_lines = file\_obj.read()
12. with open('file.txt', 'r') as file\_obj:
13. b\_lines = file\_obj.read(1)
14. c\_lines = file\_obj.readline(4).strip()
15. d\_lines = file\_obj.readlines()
17. with open('file.txt', 'r') as file\_obj:
18. **for** line **in** file\_obj:
19. e\_lines = e\_lines + line.strip() + ','
21. **print**(a\_lines)
22. **print**(b\_lines)
23. **print**(c\_lines)
24. **print**(d\_lines)
25. **print**(e\_lines[:-1])

Figure 1 - Script demonstrating various methods of reading and writing to text files – formatted at [Planet B](http://www.planetb.ca/syntax-highlight-word)[[1]](#footnote-1)

When a file is opened in write mode, I have two methods available to me. The first is write() which writes to file in a string and the second is writelines() which writes to file with a list of strings. It’s important to remember that newline characters need to be added to strings where I want line breaks in the file. There are several benefits to using text files including being cross-platform, most OS come with tools to edit them, and they are human readable.

## Binary Files

The open function also has modes for opening a file in binary for reading and writing using ‘rb’ and ‘wb,’ respectively. Binary files are beneficial for storing data when it is not required to be in user readable format and they remove the additional steps needed to format objects to a text file that require string type data. By importing the pickle module, I can serialize multiple data types in a way that it can be stored as binary information.

I can write use multiple data types including dictionaries, integers, and lists can be written to binary files with pickling. The dot notation of pickle.load() is used to read binary data and needs the file object as an argument to return the data in the format it was written. Pickle.dump is for writing to binary files and uses the data to be written as the first argument and the file object as the second.

1. **import** pickle
3. dump\_list = ['1234','5678']
5. with open('somefile.dat', 'wb') as file\_obj:
6. pickle.dump(dump\_list, file\_obj)
8. with open('somefile.dat', 'rb') as file\_obj:
9. load\_list = pickle.load(file\_obj)
11. **print**(load\_list)

Figure 2 - Script demonstrating pickle module reading and writing to binary file - formatted at [Planet B](http://www.planetb.ca/syntax-highlight-word)[[2]](#footnote-2)

I first came across [Python documentation](https://docs.python.org/3/library/pickle.html)[[3]](#footnote-3) when researching the Pickle module. I found it a chore to read not only because of the limited formatting but the heavy use of technical terms and lack of concrete examples. The [Python Wiki](https://wiki.python.org/moin/UsingPickle)[[4]](#footnote-4) is much easier to read, largely because it short and to the point, but I’m left wondering how to close a file when it’s opened in the manner shown. I enjoy [Real Python](https://realpython.com/python-pickle-module/)[[5]](#footnote-5) tutorials because of their strong formatting which makes finding information easy and their thorough explanations. There’s usually just enough there to cover our course material but it goes deeper into the topic if I want to read more. Unfortunately, some of the more involved instruction is locked.

# Error Handling

There are several error types in Python like ZeroDivisionError, when I divide by zero; ValueError, when I try a string convert to an integer but can’t; and FileNotFound, when I try to open up a file but it’s not in the directory. I can also create my own errors by making custom classes. Errors like these cause programs to crash if there aren’t statements prepared to handle them and good programs should be able to run without crashing. Python allows me to intercept errors and, instead of crashing and displaying the error to the user, I have a chance to communicate to the user what triggered the error and continue on with the program.

There are several statements associated with error handling. The first is the try statement which will run as long as there are no errors. Except statements follow try statements and only run if there are errors, in which case the try statement fails. Except statements can run in case of specific errors like a ZeroDivisionError or be general and run in the case of any error. I can also define the exception as an object which contains all the information that I would otherwise see if the program crashed and selectively display pieces to the user. Finally, the error handling end the error handling with an else statement but it only runs if the try statement succeeds. By using the raise keyword along with an exception, I can jump straight to the specific exception statement in an error handling code block.

1. **class** LessThanZero(Exception):
2. """Value is less than zero"""
3. **def** \_\_str\_\_(self):
4. **return** 'Divisor value is less than zero'
6. dvnd\_num = 20
7. **print**('Type \'exit\' at any time to exit.')
8. **print**('The dividend is 20.')
10. **while** True:
11. **try**:
12. dsor\_str = input('Give me a divisor.\n')
13. **if** dsor\_str == 'exit':
14. **break**
15. dsor\_num = int(dsor\_str)
16. **if** dsor\_num < 0:
17. **raise** LessThanZero()
18. q\_num = dvnd\_num/dsor\_num
19. **except** ZeroDivisionError as e:
20. **print**('Cannot divide by zero.')
21. **print**(type(e))
22. **except** LessThanZero as e:
23. **print**('Please enter a number greater than zero.')
24. **print**(e.\_\_doc\_\_)
25. **except** Exception as e:
26. **print**('Maybe try entering a number...')
27. **print**(e)
28. **else**:
29. **print**('The quotient is', q\_num)

Figure 3 - Script demonstrating structured error handling - formatted at [Planet B](http://www.planetb.ca/syntax-highlight-word)[[6]](#footnote-6)

The explanation of error handling at [Geeks for Geeks](https://www.geeksforgeeks.org/python-exception-handling/)[[7]](#footnote-7) is really clear and builds on the concept with simple examples followed up by a video for anyone who learns better through demonstration. I may find the documentation for Python to be bland and difficult to read but the [documentation tutorials](https://docs.python.org/3/tutorial/errors.html)[[8]](#footnote-8) are well-organized and easy-to-follow. The numbered table of contents is a nice feature if I want to skip over a specific topic. Because it’s also the main Python site, I feel like I can trust it more than an answer I might look for on a lesser-known blog.

# Assignment Seven

The first step I took to modify this week’s script was to import the pickle module in order to read and write binary data. I then changed strFileName to a binary data file ending with .dat instead of a text file ending in .txt. Next, I needed to change the read and write functions in the FileProcessor classes to use the imported pickle module rather than the read and write methods for text files.

Graphical user interface, text

Description automatically generated

Figure 4 - CDInventory.py running in Spyder

The read\_file function has changed to rely on a single argument which is the file name of the file being read. The file is then read in binary mode with the pickle.load method and the data is assigned to the inventory variable which is then returned so that it can be assigned to the table in the main program. The write\_file function still relies on the file\_name and table arguments but is reduced to two lines of code which uses the pickle.dump method to write the table to file.

I first added error handling on program start when the program tries to use the read\_file function and assign it to lstTbl. If there were no file by the name of CDInventory.dat in the directory then the program would crash so the except statement will display to the user that there is no file to load. The else statement will print to the user that the file was loaded to data once the try statement succeeds in assigning the return value of the read\_file function to the data table. Similar error handling is also required when the user chooses to load data to inventory from the main menu. The program tries to assign the contents of the file to the table but if there is an error the user will be told that there is no CD inventory to load. Upon successfully loading the file to data, the program prints that the file was found and uses the show\_inventory function to display it.

Text

Description automatically generated

Figure 5 - CDInventory.py running in terminal

Both the del\_choice and input\_inventory functions in the IO class take string inputs from the user and attempt to change them in to integer types so I’ve added error handling for each. The program will try to use the integer function on the string value and if there’s an error the function will return a value of None. If the conversion from string to integer succeeds, then the integer values will be returned for use in the main program loop. If None values are returned then the program prints that the value must be an integer and continues to the main loop otherwise if there are integer values, the program uses them either delete an ID or store an ID from data, depending on the function.

# Summary

This week I’ve learned that I can read and write both simple text and binary data to files and error handling is a useful way to prevent programs from crashing. The big downside for text files is that Python only stores strings in text files. The benefits of binary files are immediately apparent in the assignment when I can reduce my code by almost ten lines just by not having to format data into strings. The only downside to binary files seems to be that the pickle module is specific to Python.

Exception handling is an elegant solution that I find comes easily to me because they are similar to if statements at their simplest. However, they can also be incredibly involved and provide me with a wide array of ways to help make the user experience better. I’m glad I no longer have to just rely on if statements to prevent program-crashing errors.

After the seventh week of course materials, I realize I’m very much used to tutorial-style learning but I can see a situation where I’m not always going to find the answer on a blog or a Youtube video. Challenging myself to read Python documentation will help me grow as a programmer by getting insight from the source that a second-hand account might not have.

1. Captured 2021-Feb-28 [↑](#footnote-ref-1)
2. Captured 2021-Feb-28 [↑](#footnote-ref-2)
3. https://docs.python.org/3/library/pickle.html [↑](#footnote-ref-3)
4. https://wiki.python.org/moin/UsingPickle [↑](#footnote-ref-4)
5. https://realpython.com/python-pickle-module/ [↑](#footnote-ref-5)
6. Captured 2021-Feb-28 [↑](#footnote-ref-6)
7. https://www.geeksforgeeks.org/python-exception-handling/ [↑](#footnote-ref-7)
8. https://docs.python.org/3/tutorial/errors.html [↑](#footnote-ref-8)