# Introduction

For week 7 we were given the task of modifying code from assignment 6. These modifications were to include adding structured error handling to areas in the script where user interaction, type casting or file access could cause errors. In addition to including error handling, we were tasked with changing the permanent data store to binary. Below are the detailed steps taken to complete the project.

# Assignment 07

A foreword: this by far has been the most challenging assignment to date; one in which I spent a majority of time in labs attempting to understand code structure. And honestly, I’m still a bit conscientious about error handling; especially when interacting with functions. Regardless, it’s better to attempt and fail then to not try at all. The first step taken to begin this assignment was applying changes given back by the instructor.

Once the initial program had been updated with corrections to improve functionality I began work, once more, modifying the script. I began running and re-running the program to see where I could intentionally cause an error. It didn’t take long to crash the program. Two errors which were immediate were ValueErrors when a user would attempt to:

1. Add a cd
2. Delete a row from the .txt file

**Fig 1 - ValueError Message - empty argument passed in Spyder**

![Text

Description automatically generated]()In short, when a user attempts to add a cd to the .txt file they’re prompted with an ID number, after pressing “a”; the cursor can’t be left empty or as a string statement when ‘enter’ is pressed. Similarly, for the updated script which deletes rows (when prompted); a user can’t pass string letters / statements or pass empty arguments – the delete\_row function only accepts numerical values, returning an error message if the wrong number is entered.

For exception handling I wanted to try and allow the script to pass either a string statement or a none type when adding a cd to the output file without throwing a fit. Likewise, I wanted to provide exception handling to the delete function as well so the program wouldn’t shut down if god forbid a user accidentally hit enter when an argument hadn’t been passed.

By doing this I would be adding structured error handling where user interaction as well as type casting (string to int) and file access operations wouldn’t crash the program (at least in theory). However, before I could start on that, I wanted to update the existing script so it would pass all user inputted data as binary to an outbound data file.

To do this I imported the pickle module in Python, changed the .txt file to .dat file and began fiddling with the read and write functions in the script. To assist with this I had access to a good resource in the GeekforGeeks about section on pickling (2018) the link for which is provided in the references section. A couple of issues I had to deal with pickling is that I had to comment out FileProcessor.read\_file(strFileName, lstTbl). The reason being is when my program would start up it would immediately crash because the .dat file I was using would have binary code already stored and this function would just stop the entire program.

To prevent this, that specific function was commented out – at least until the read and write functions were updated with pickle functionality. Then it seemed fine so I commented it back in.

The next issue was trying to convert str to byte. Thankfully, I received some last-minute insight from the instructor that the pickling module doesn’t need to worry about conversions (the code in the functions did however) and because of that and will instead take what’s provided and reload it without hassle. However, an issue I’m encountering with this (and I do not know why) when a user specifies they want to load the stored data the program doesn’t open the binary file and instead only states: loading . . .

Whether this is an issue with Python performance, or I just inputted the wrong script (likely the latter), I couldn’t tell you. What I do know however, is the program is not crashing and still saves inputs into the .dat file when prompted as binary. And as the instruction for the project were only to store the data in this way I can at very least claim to’ve met that bare minimum. With the script updated and pickling (more or less) “working”, I began to focus on error-handling.

I focused on what I believed to be the easier task – fixing the delete functionality. To do this, I simply inputted the following code:

|  |
| --- |
| intIDDel = ‘’  try:  intIDDel = IO.del\_row\_input()  except ValueError:  print('Please enter a number value') |

First, I had to define intIDDel as empty string. Since this variable is called in a separate function in my code, it automatically crashes the program so defining the value, came first. Once I defined intIDDel, I used try, except logic, basically stating: if there’s a ValueError call, when a user inputs something other than an integer to intIDDel, raise the error message: ‘Please enter an integer value’. So now, when a user is prompted in the program to delete a row from the lstTbl and accidentally hits return or inputs a value other than an integer, they’re greeted by the print statement above.

With that done, I moved on to fixing the ‘add cd’ error. This was a bit cumbersome to pinpoint which variable the program was using to generate the ValueError message. Eventually I located the root variable in my add\_Row function. Here, I wanted to provide an exception to allow the user to accidentally press return or a string without crashing the program. To do that I inputted this code:

|  |
| --- |
| int\_id = ‘’  try:  int\_id = int(str\_id)  except ValueError: |

It doesn’t look like much (especially without the error message I created) however, the int\_id, which is set to empty string and then cast to variable int(str\_id) was the biggest part of making this exception handling code work. I didn’t list it here but that int\_id was set to the [0] list index then given user input in a dictionary row. Setting it to empty string and providing an exception made it, not only print a warning message and prevented the program from crashing, but as a fun little accident, now when a user has accidentally passed string or a none-bject into the ID: prompt – if they enter: ‘d’, to delete a row, they can now pass a none argument or string to delete the corresponding row from the table. Maybe. It’s not crashing the system and it IS functioning as it’s supposed to so I can accept it as is.

The last bit of error exception handling I did for my code was place an exception for the function FileProcessor.read\_file(strFileName, lstTbl). That code looks like this:

|  |
| --- |
| try:  FileProcessor.read\_file(strFileName, lstTbl)  except Exception:  pass |

The reason for this was quite simple; I wasn’t quite able to get the pickle.load functionality of the program down and every time the program booted up it immediately read what had been stored in the file previously. To “fix” that, I added this exception pass functionality to allow the program to run, even though there was nothing stored in the binary .dat file. Must’ve worked because here is my program running in both spyder and CMD.

![Graphical user interface, text

Description automatically generated]()

**Fig 2 - Spyder Capture of Code Working**

**![Text

Description automatically generated]()**

**Fig 3 - Code working in CMD**

# Conclusion

This was one of the harder assignments. Fortunately, there were great resources available in helping me along with understanding the concept of pickling (of which I need to improve) and error processing. I previously mentioned GeekforGeeks but another a great resource was kite(na) which provided a bit of the exception, pass code logic that helped me maintain the functionality of the program given its issue with reading the pickled .dat file. However, one thing I’ll say about these resources is some are very hit or miss in terms of keeping my attention. It’s evident no one is trying to hoard their knowledge, but at the same time, trying to understand that knowledge can be hard to do as a beginning. Especially when all that information is coming at you all at once. As a, for instance, a website called Programiz, I don’t doubt is helpful to anyone with already in depth knowledge of Python, but for someone who has spent six hours attempting to get code to run properly – it’s a hurdle to ingest the information they’re trying to help you with. And with that, I’m done and looking forward to being one more week closer to the end of the course.

# References

Understanding Python Pickling with example (2018), <https://www.geeksforgeeks.org/understanding-python-pickling-example/>

Kite (na) <https://www.kite.com/python/answers/how-to-ignore-an-exception-in-python>

# Index

Fig 1 - ValueError Message - empty argument passed in Spyder

Fig 2 - Spyder Capture of Code Working

Fig 3 - Code working in CMD