Introduction

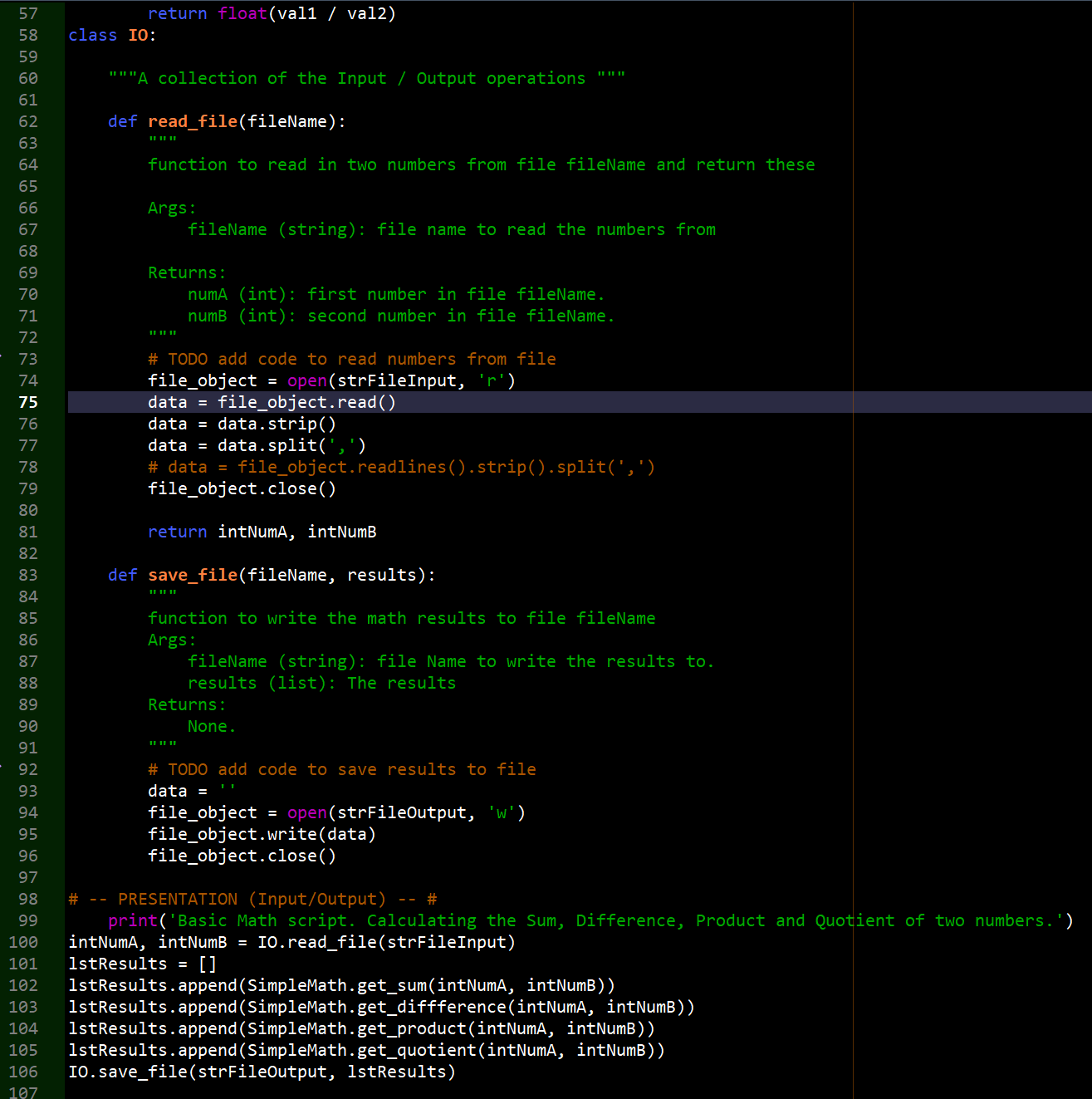
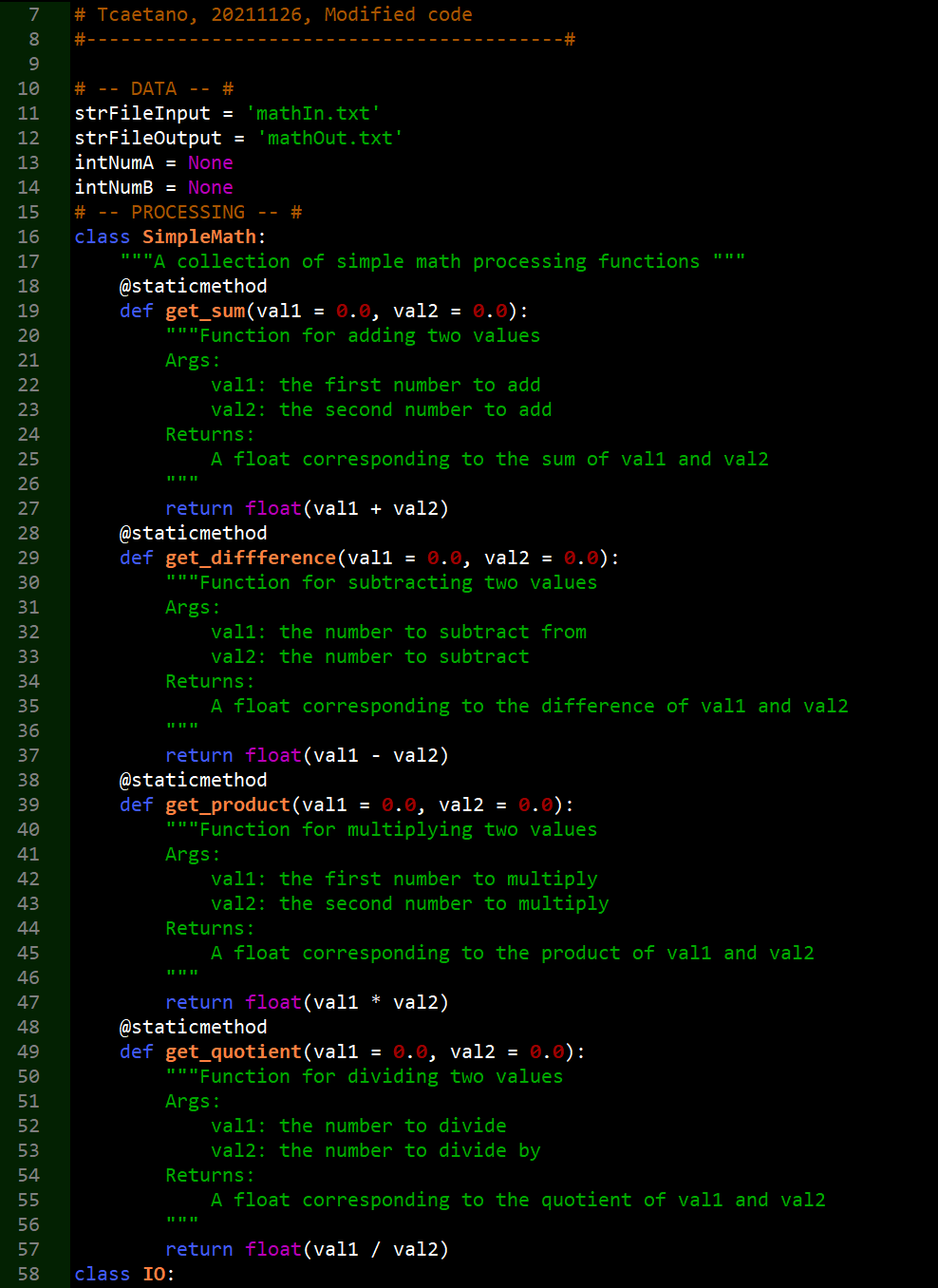
Assignment VI consisted of three LAB lessons and one extensive section of code that used the concepts of classes, functions, pickling and error handling.

The concepts of pickling (saving and retrieving files stored in the binary format) and error handling by means of the TRY and EXCEPT commands were introduced and used in the labs.

The information below details the paths I followed in an attempt to understand and grasp these concepts.

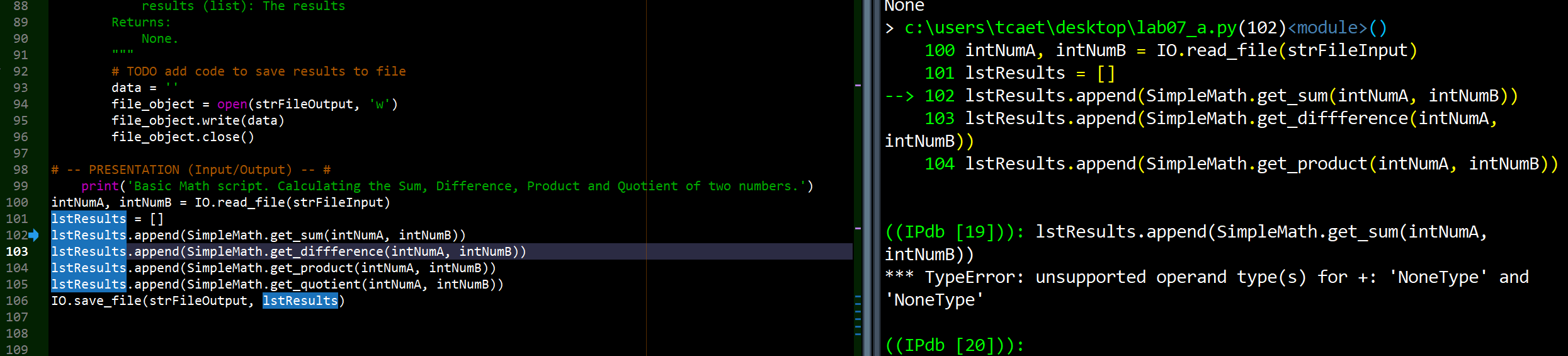
GitHub link:

LAB 07-A: Working with text files  
In this Lab, we’ll continue with the code form Lab 06-C (I have included a sample solution in this Module’s zip file. You’ll  
create a class for I/O and we’ll replace the user input with an input file that has one line with two numbers in it,  
separated by comma. We’ll replace the print output with writing to a file, all four results on one line, separated by  
commas. Do not change the existing code in the SimpleMath class!  
• Make a copy of the LAB07\_A\_Starter.py file and save it as LAB07\_A. This file has already the class IO and  
function docstring prepared.  
• Add code at the first TODO to read the numbers from a text file  
• Add code at the second TODO to write the result to a text file.  
• Test the script and write down how the code works.  
• Think about what needed to be changed to handle multiple lines in the input file with two numbers each. (And  
write the results of the calculations out to the Output file!)

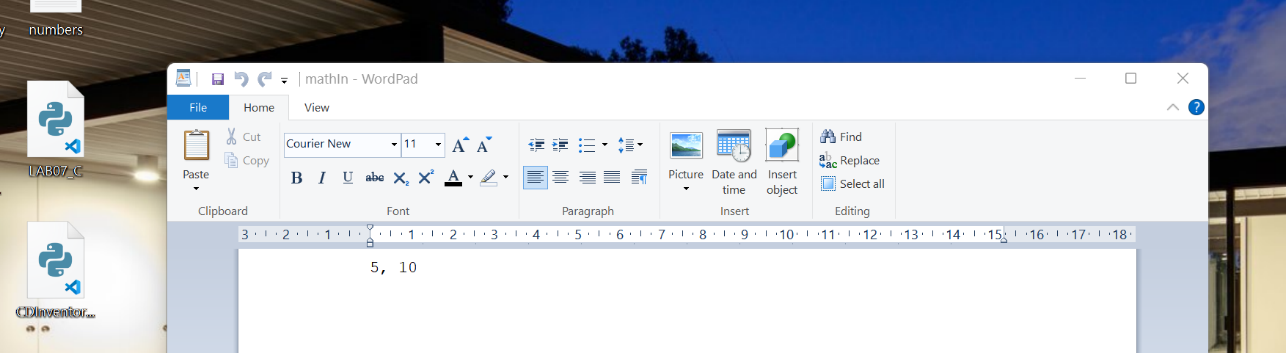


Here is the full code for this assignment. I feel that I was pretty close to getting this fixed but ran out of time trying to make it fully function. I see the that is an issue with a variable not being defined.

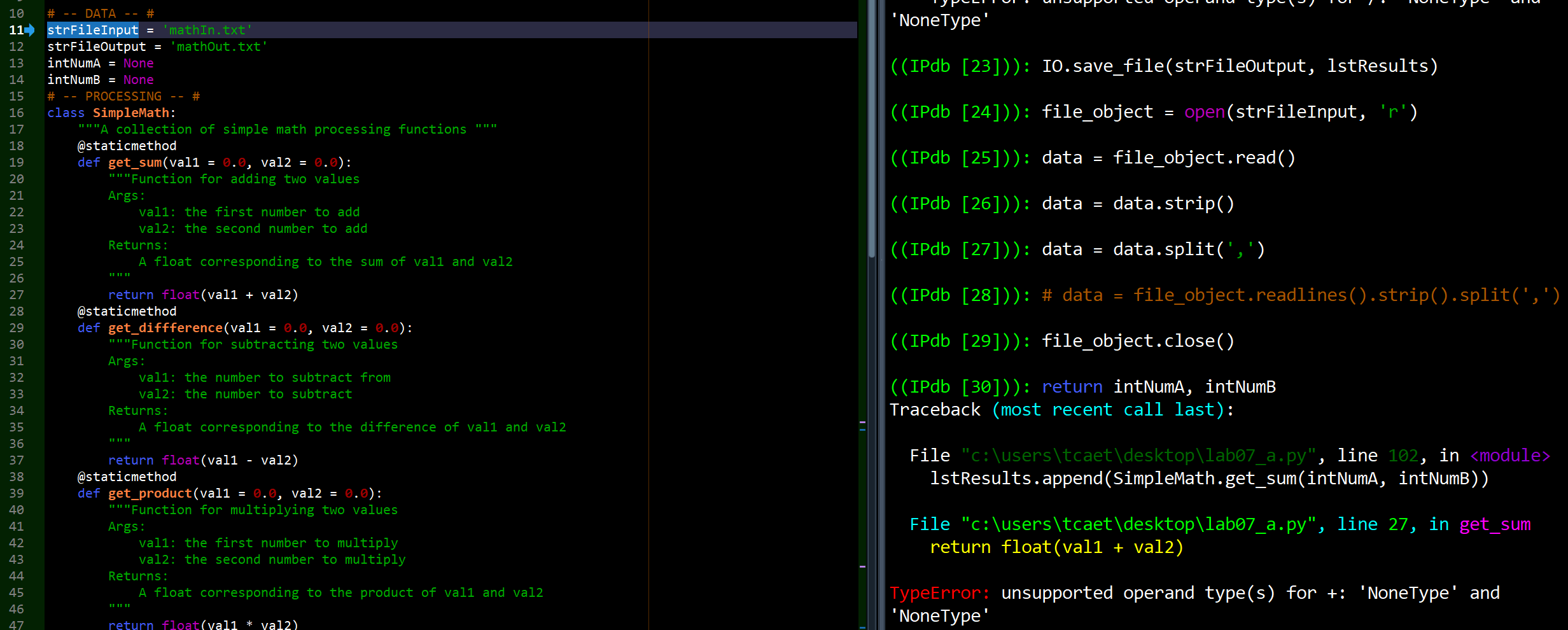
Here is some of the troubleshooting I executed:



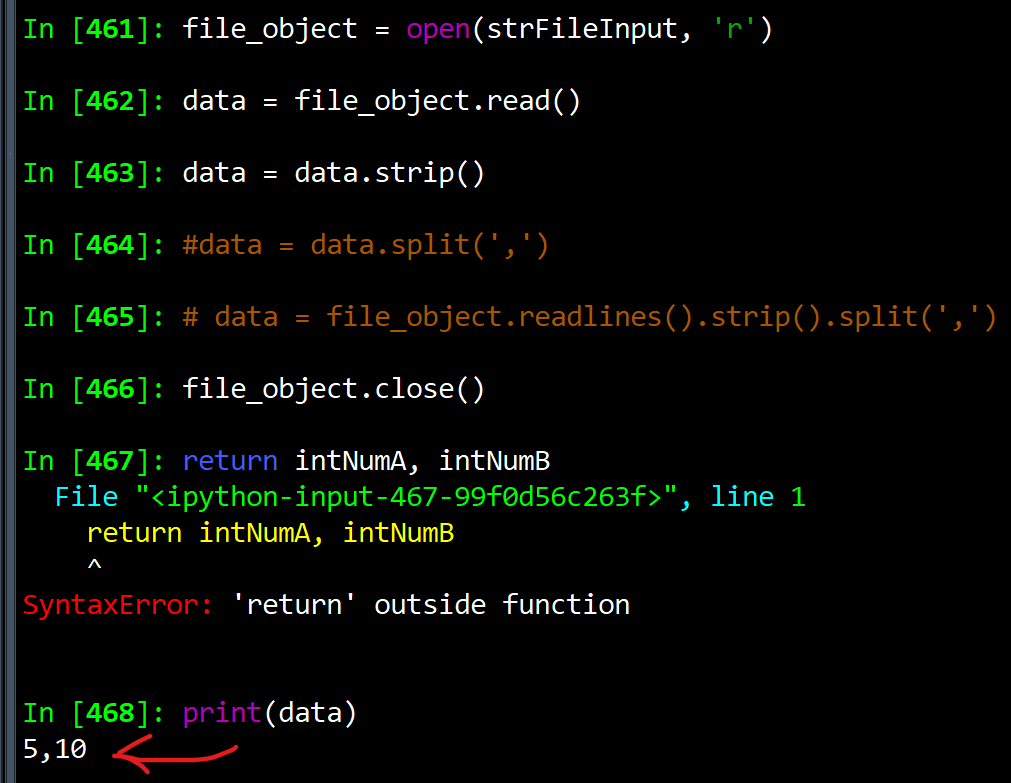
I see that there seems to be a disconnect here with the intNumA and intNumB variables as these have None values assigned initially. I tried without these values but got the same result.



The mathIn.txt file was created initially and set to the correct path in the Desktop – Spyder also points to that same path.



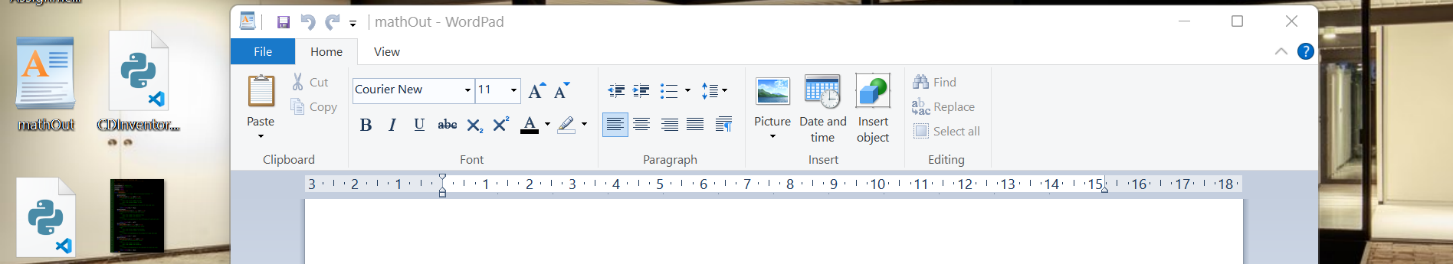
Cycling through the code, it seems that the file reading and writing portion both work.



The file is being read as shown – printing(data) returns the values inside the mathIn.txt file

An issue I am having is that yesterday I was able to return 5 for IntNumA and 10 for intNumB, but today I am not able to make it happen.

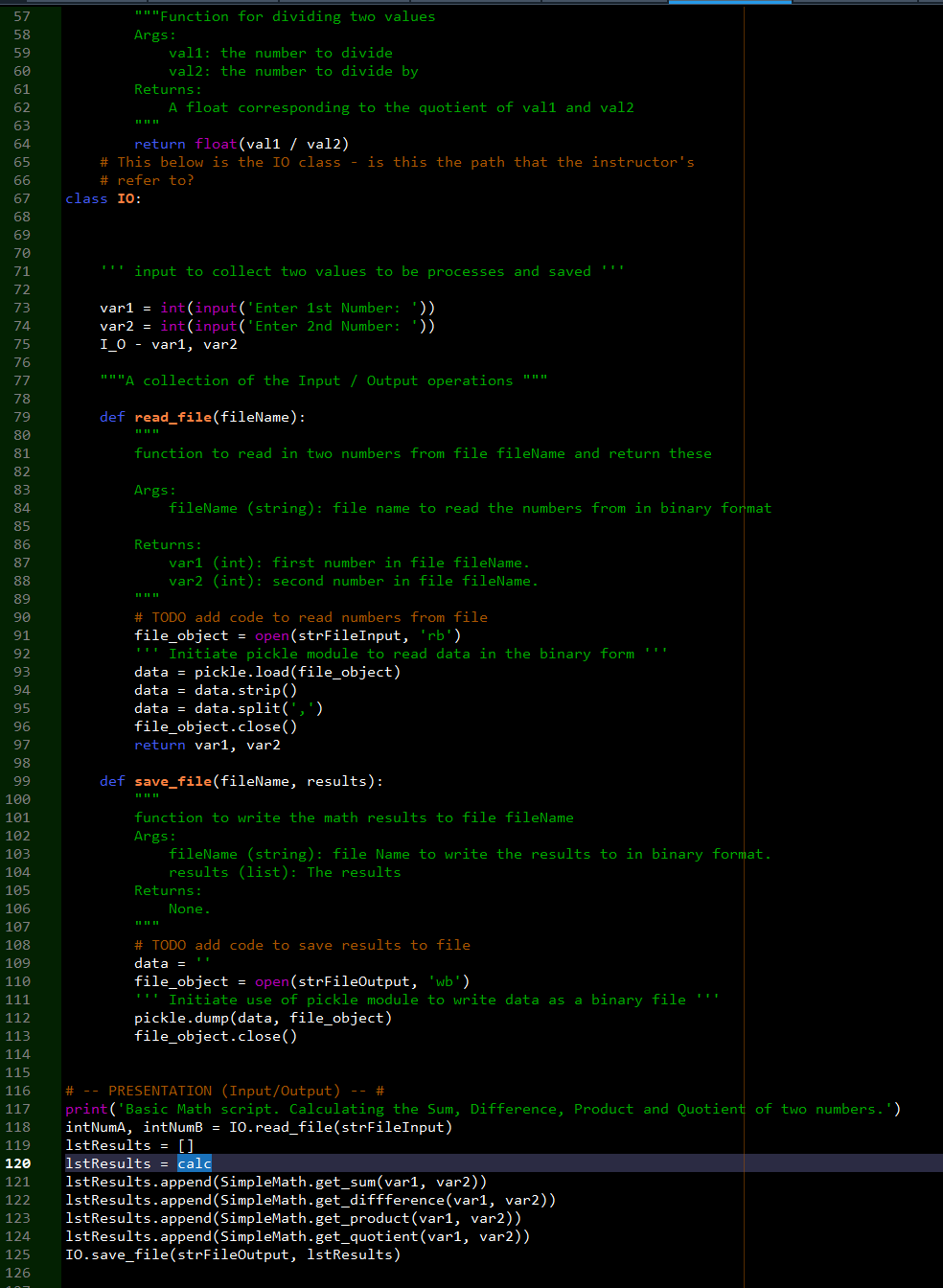
The error ‘return’ outside function continues to appear.

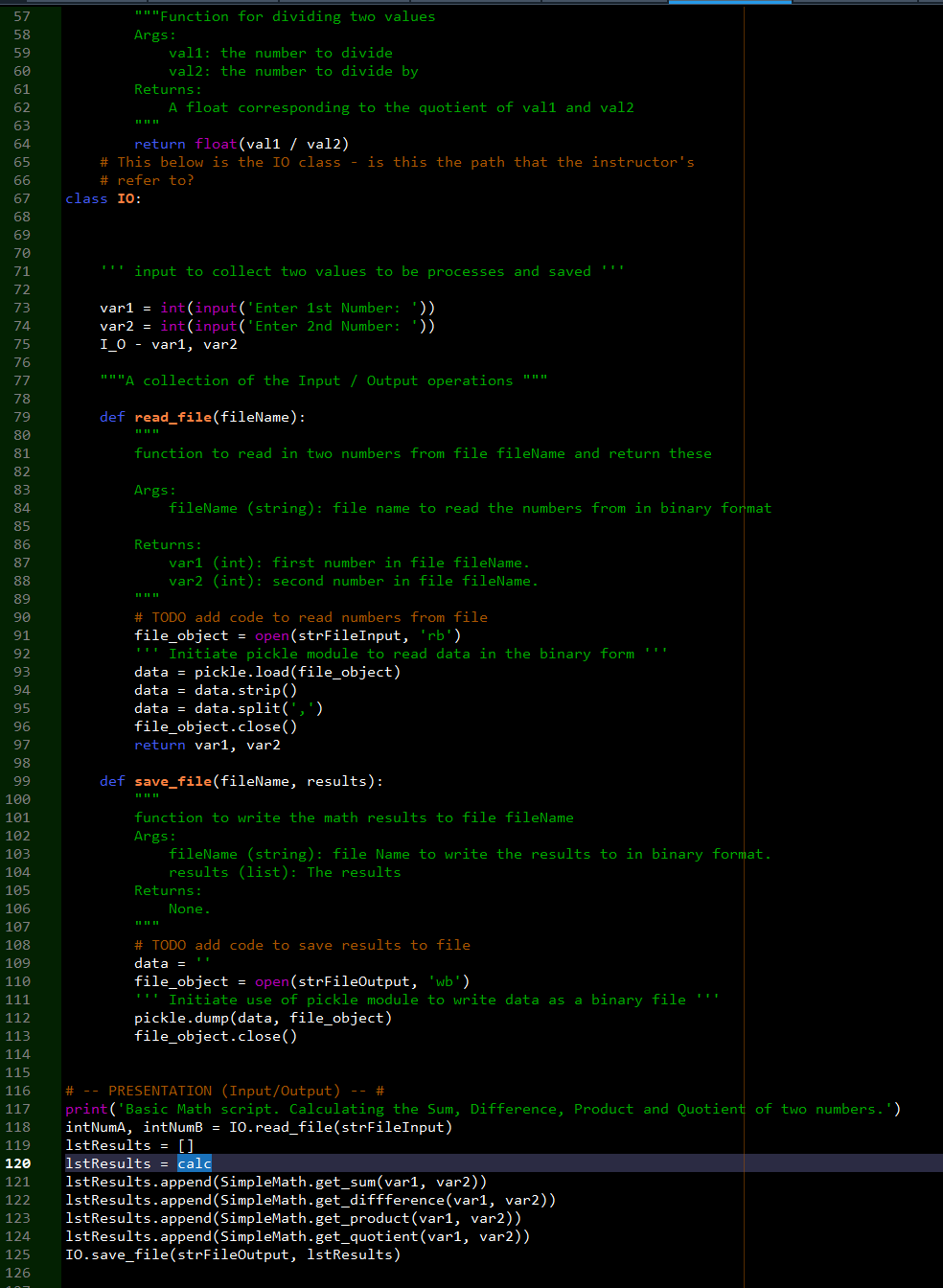


The code that creates the mathOut.txt is also working as shown here, the file is created onto the desktop, albeit without anything inside it.

How does the code work? It reads file input from an external file. It loads it into the code and the functions which have rules defined by four mathematical operations then perform these calculations and save the values onto a file which is then exported onto an external medium. The results are calculated outside the functions by use of the CLASS which summarises all the commands saved within the functions. It was actually fairly easy to comprehend but difficult to execute

LAB 07-B: Working with Binary Files  
In this Lab, we’ll continue with our SimpleMath program: You’ll change the file input / output functions to work with  
binary files. Additionally, you’ll add the functionality to select the operation mode via program arguments (see Module  
03): Either processing the calculation or displaying the results and user input for the numbers used in the next  
calculation  
• Make a copy of the LAB07\_A.py file and save it as LAB07\_B.py.  
• Add code to read in the program arguments and create distinct operational paths: As program arguments we’ll  
use ‘calc’ and ‘IO’.  
• Change the functions for the file access to use binary files. Change the names of the files to numbers.dat for the  
numbers file and results.dat for the results file.  
• For the ‘IO’ path, add code to display the current content of the results file. Ask the user for input two numbers  
and save these to file.  
• For the ‘calc’ path, read in the numbers from files, do the calculations and write out the results into file.  
• Test the script and write down how the code works.  
• Think about what this is useful for?





Here is the code for LAB07\_B, I went through lesson three but could not find a way to pass these arguments as requested. I spent more than a day trying to figure out

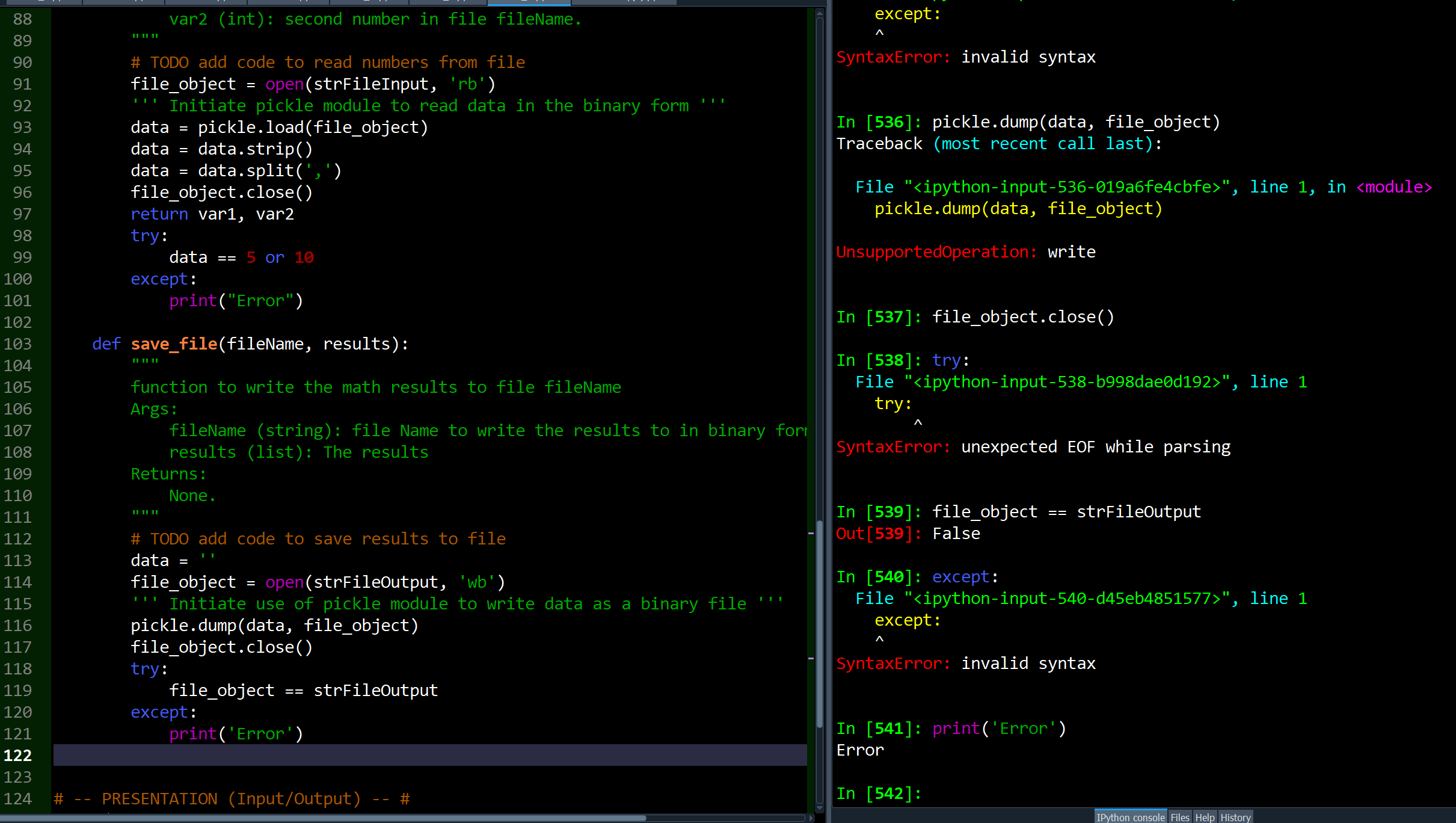
how to convert the IO CLASS into a path. It turns out that IO was not supposed to be a CLASS although a CLASS entitled IO already existed. When you capitalise the CALSS

IO and then remark something to the effect of using a path named IO and calc, it is confusing as I was under the impression that a capitalised CLASS named IO would have

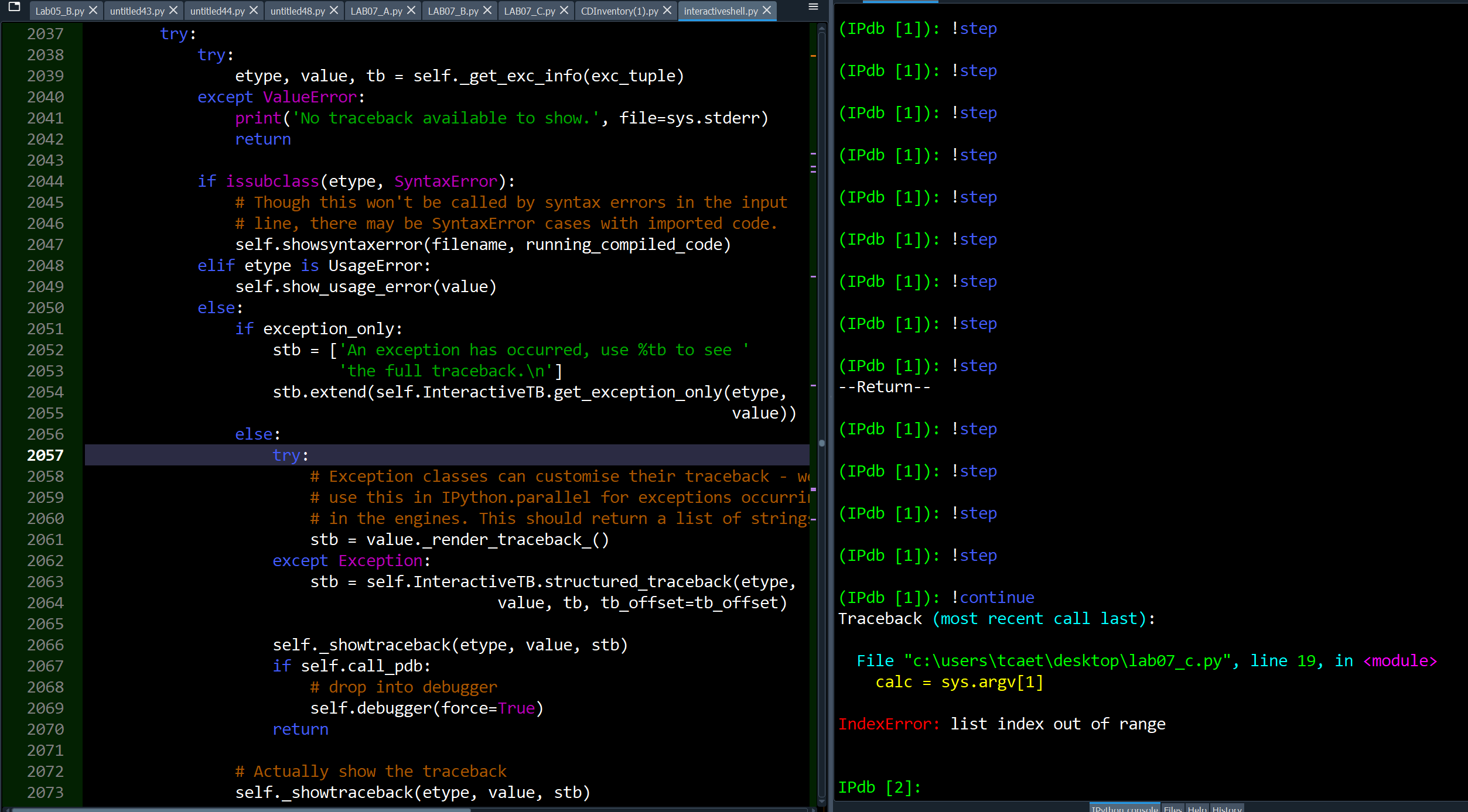
the exclusivity of the name IO. If you want to refer to something related to input and output why not call it Input/Output, I/O, I\_O, etc? This way the CLASS IO will be left alone. Or perhaps one could mention that the path to IO is something separate from the CLASS IO? Anyway, enough chin wag. This was simply confusing and I spent entirely too much time trying to understand the instructions.

I imported the module SYS into the code, created two argv’s entitled calc and IO and tried to make this work but ran out of time at 17h30 when it is time to move on to lab C and the CDInventory assignment.

LAB 07-C: Structured Error Handling:  
In this Lab, we’ll continue with our code from Lab 07-B. You’ll add error handling whenever there is an IO interaction.  
You do not need to create your own classes.  
• Make a copy of the script from LAB 07-B and save it as Lab07\_C.  
• Add code to catch errors around file operations (2 areas)  
• Add code to catch errors around the user input operations. Handle non-numeric inputs and value zero input for  
the number used as denominator separately.  
• Test the script and write down how the code works.



I understood the concept of file error handling although not too sure of why we would have to incorporate this into code given that an IDE like Spyder handles code errors. For example the interactive shell in spyder does a lot of the work although I am not sure how to read all of this:

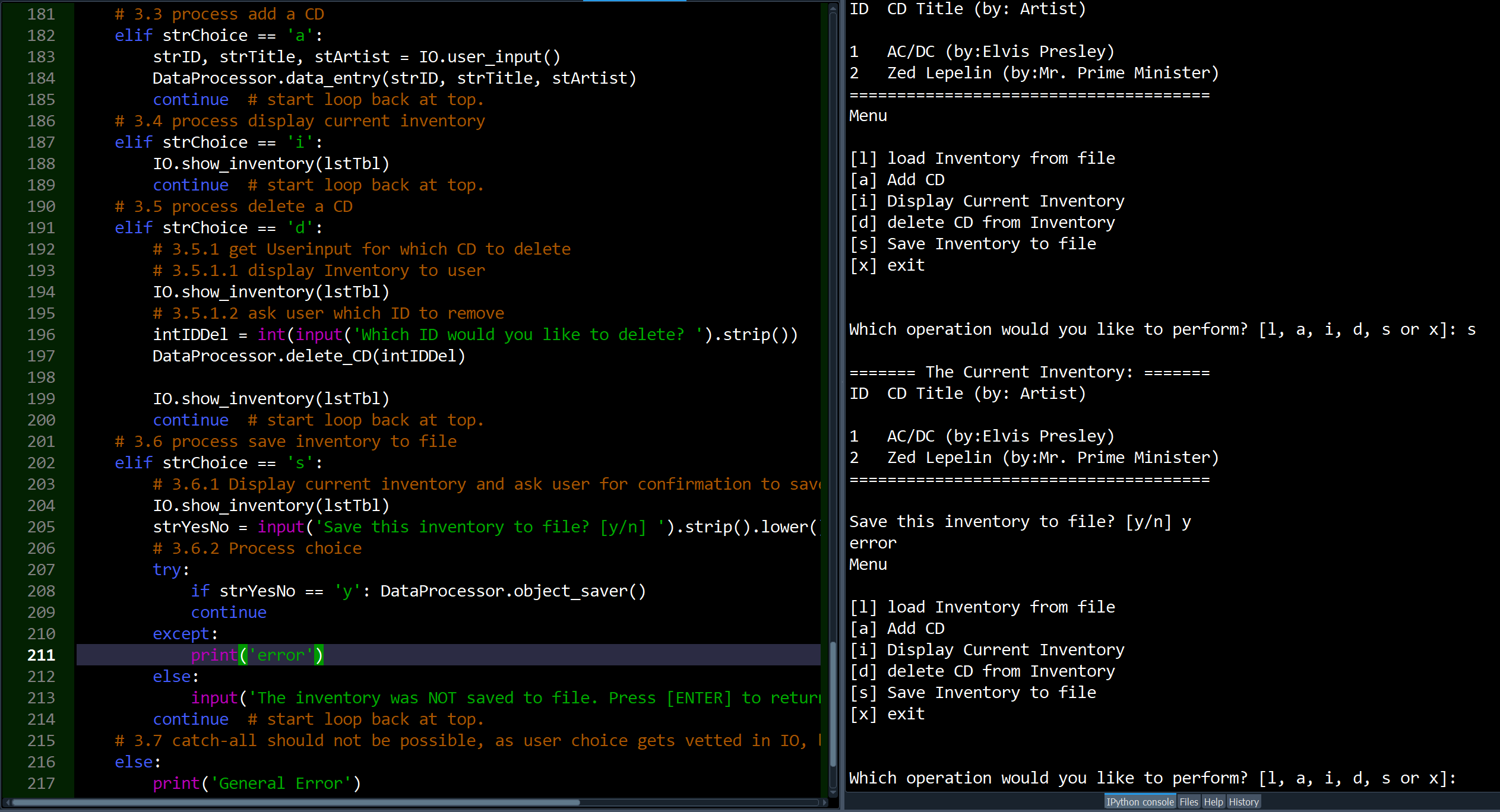


Either way, it’s difficult to make this work once your initial code is already compromised, that is errors stemming from LAB A and B have not been corrected and now I have to insert error handling into code that is already initially compromised.

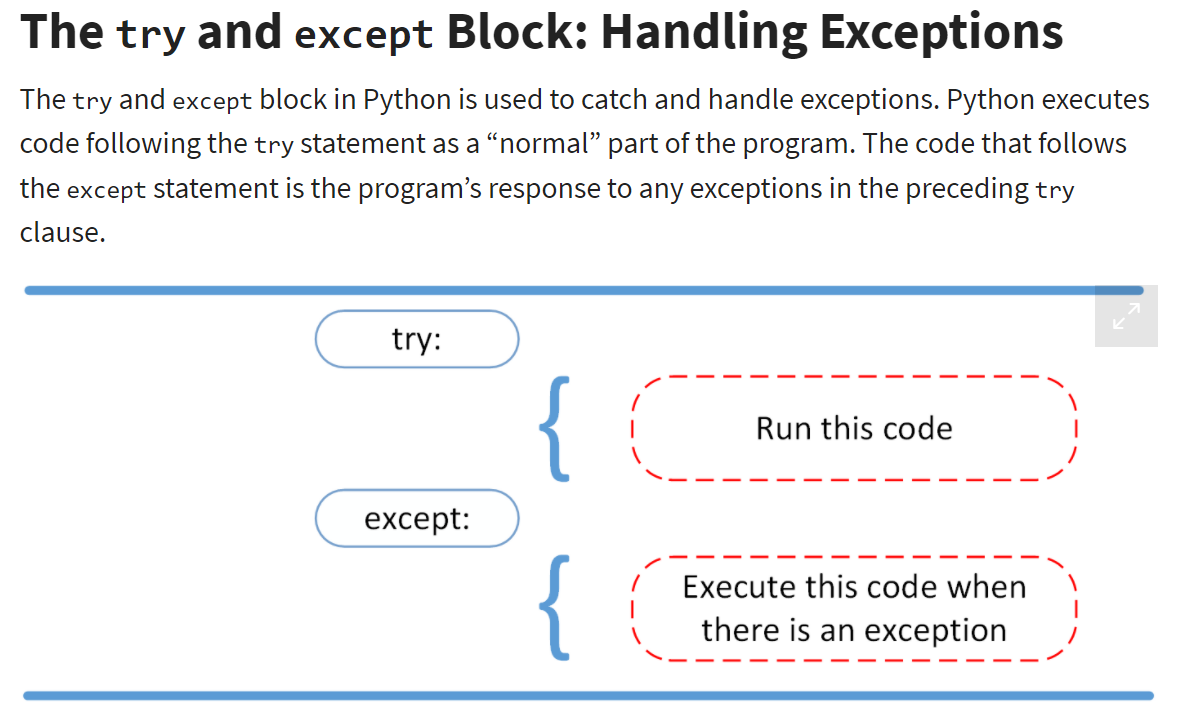
I attempted to use the TRY and EXCEPT although I was not successful in making it work, nevertheless, after reading the literature on this on the interwebs (which were invented by Al Gore), I am beginning to understand the functionality but I am left with even more questions.

**ASSIGNMENT**:

CDInventory.py



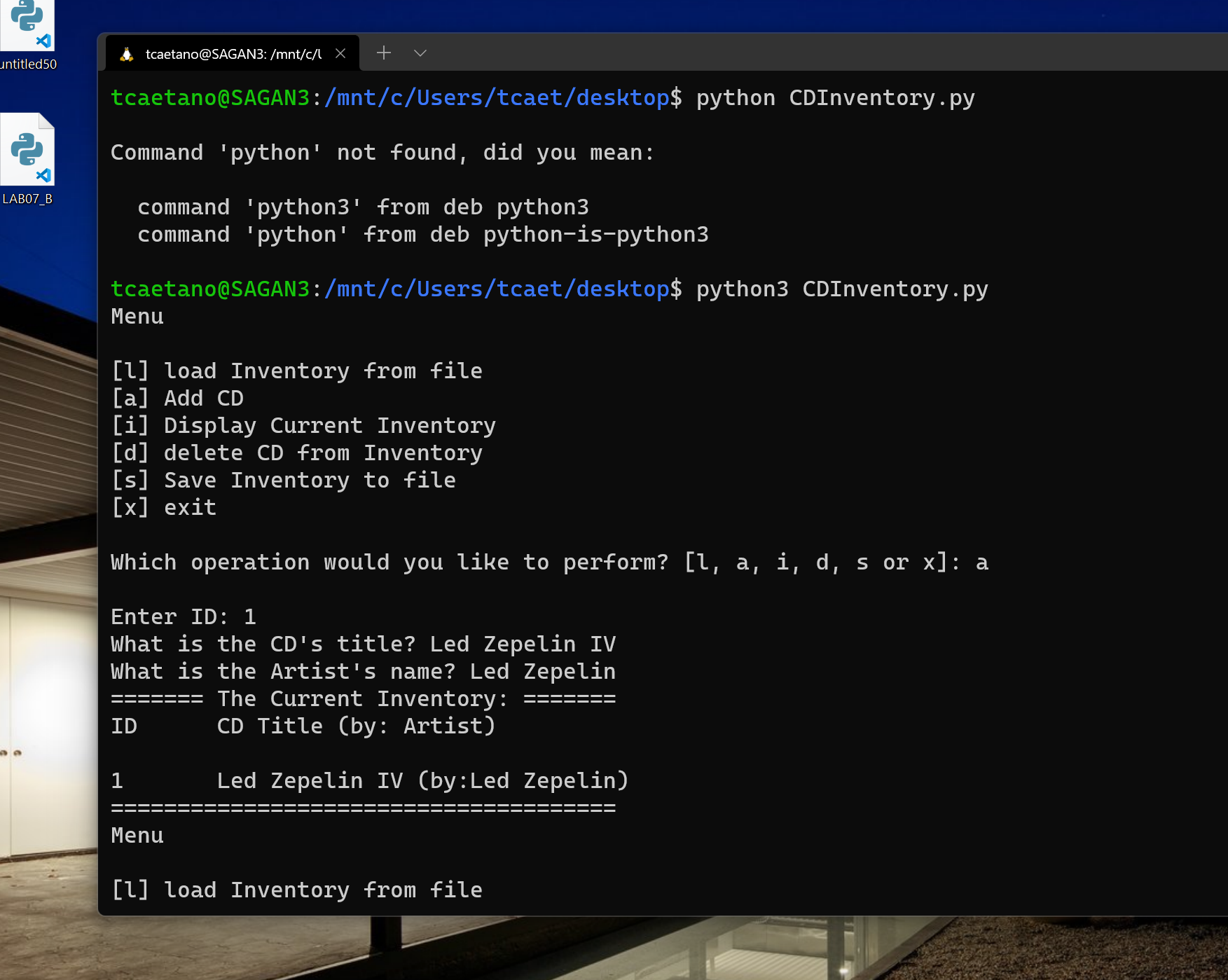
This code works but not without issues. It saves files and you can add but the formatting is not as good as when it did not have the save as binary option. I am still trying to understand how the error handling works.



Some information on the internet helps – for example, this graphical representation above, does a good job at explaining what the TRY and EXCEPT commands actually do. The word except doesn’t match what the etymology of the words means in the English sense of the word – this is programming after all, but the BASIC programming language despite its downfalls, was far more oriented towards the etymology of the English language, hence my confusion. EXCEPT thus means that whatever follows should be executed if there is an exception – this was obviously devised by someone from Holland 😊.

Despite this small obstacle, I will continue attempting to understand how this construct of catching errors works.

Another challenge I have encountered is the .SYS module and the use of arguments ARGV[]. This is still by far the biggest obstacle I have come across. Back to the drawing board I go.



From the terminal window I get some functionality. The .DAT is saved but I can’t get any pickles inside it yet.