Zach DeVries
Feb 22, 2023
IT FDN 110 A
Assignment 06

Lists and Dictionaries

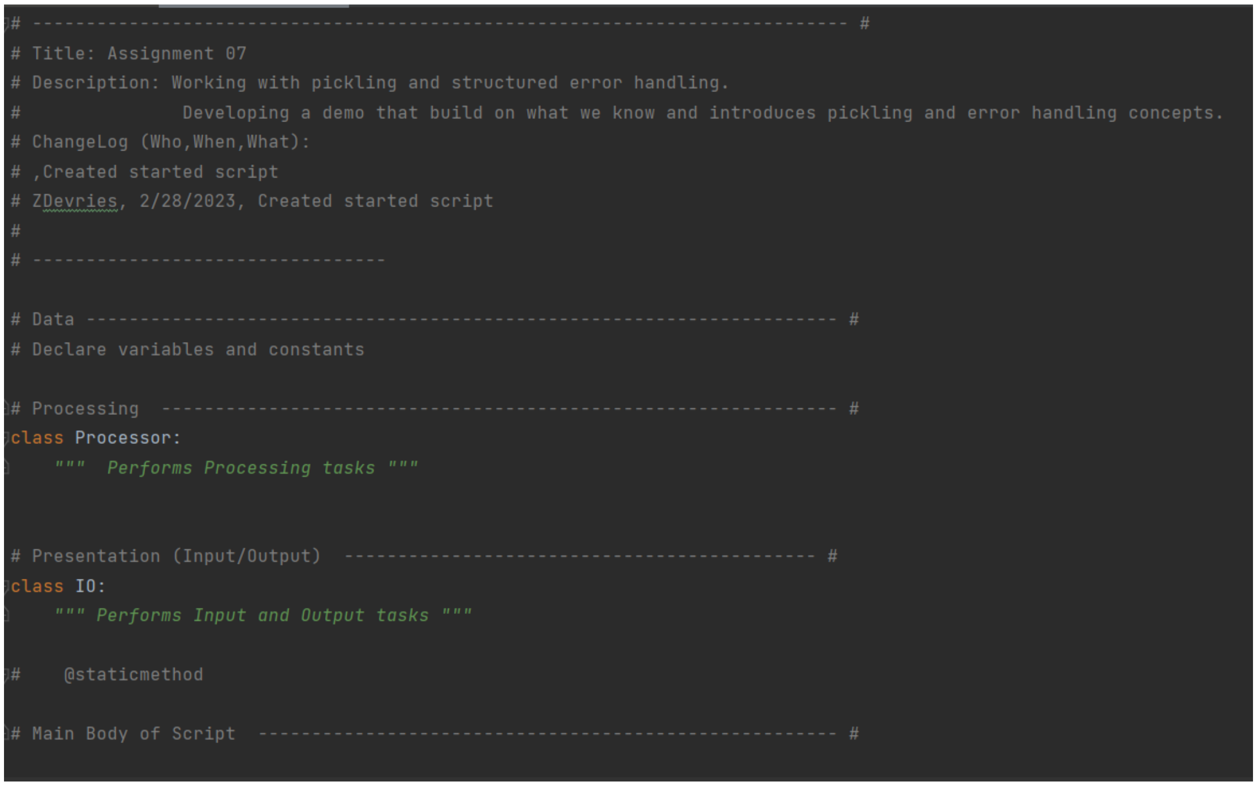
Introduction:

As part of this week's module, I learned about pickling, shelving, and error hanfling using try/except clauses.

Program:

I started off by stubbing out the program with a Header, Data, Processing, IO, and Main sections. I thought through how I wanted to demonstrate the required skills and got ready to start adding the code.

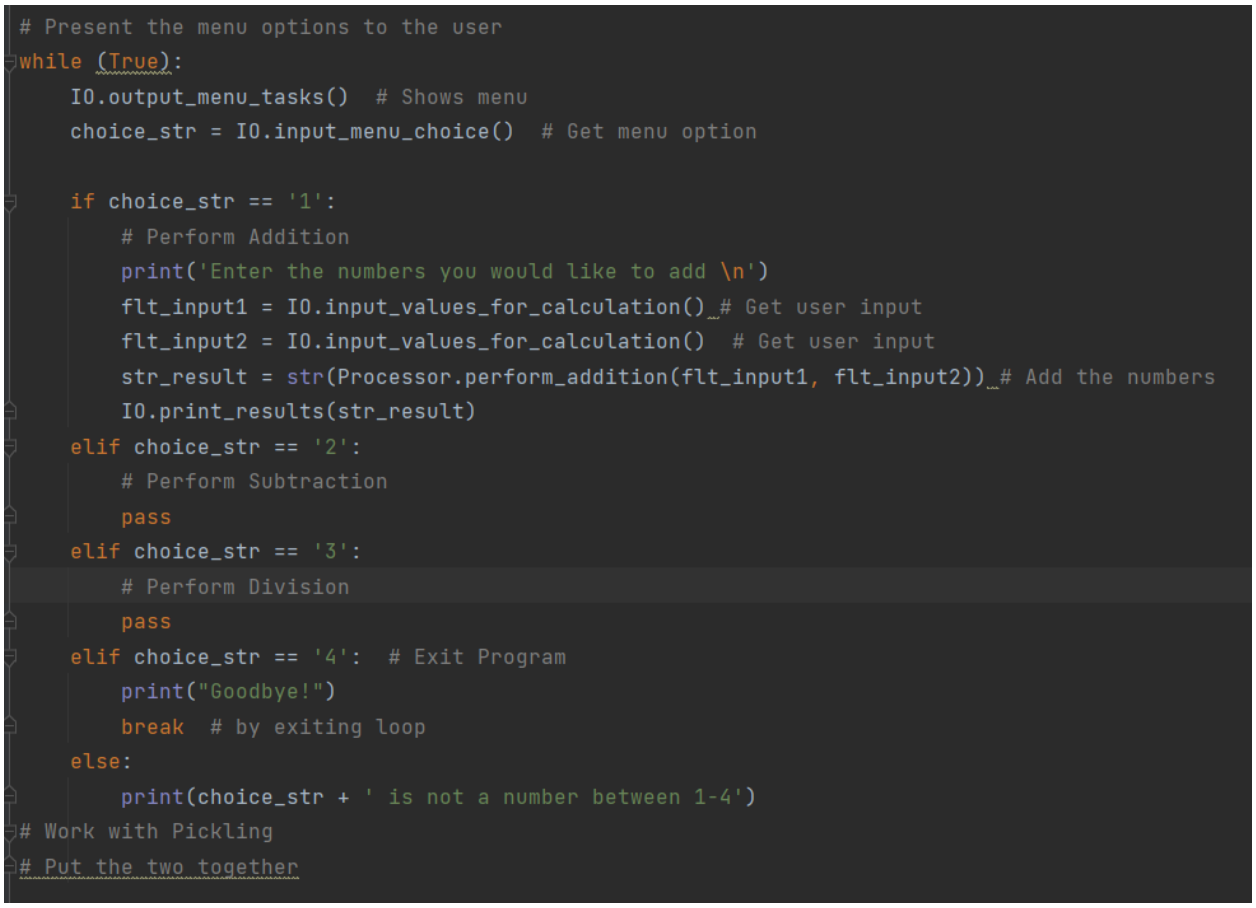
Figure 1: Program Outline



```
# ---------------------------------------------------------------------------- #
# Title: Assignment 07
# Description: Working with pickling and structured error handling.
#               Developing a demo that build on what we know and introduces pickling and error handling concepts.
# ChangeLog (Who,When,What):
# ,Created started script
# ZDevries, 2/28/2023, Created started script
#
# ---------------------------------

# Data ---------------------------------------------------------------------- #
# Declare variables and constants

# Processing  --------------------------------------------------------------- #
class Processor:
    """  Performs Processing tasks """


# Presentation (Input/Output)  -------------------------------------------- #
class IO:
    """ Performs Input and Output tasks """

#    @staticmethod

# Main Body of Script  ------------------------------------------------------ #
```

I set out on a plan to develop a calculator that can perform a few basic math operations. I planned to add in some error handling and then provide the option to save the data in the form of a binary data file. I started out with creating the functions required to perform addition. I set up a while loop to run the program and menu/user input functions to capture the desired math function from the user. Then I developed a number input and addition function to get the basic addition portion of the program put together.

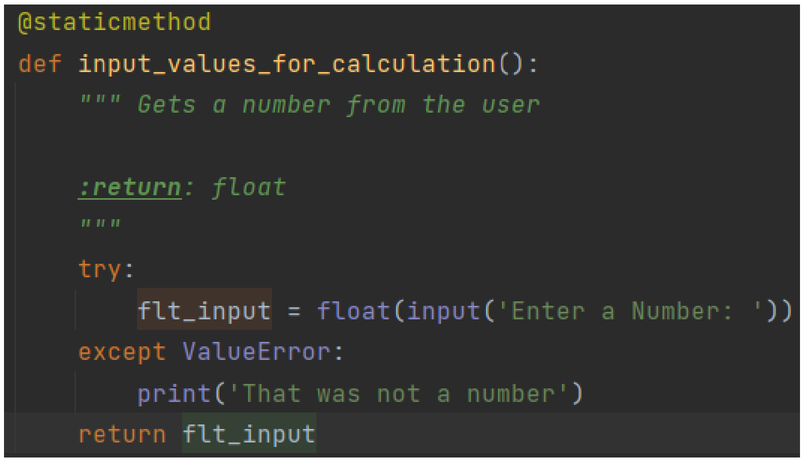
Figure 2: Main Body_Addition



```
# Present the menu options to the user
while (True):
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    if choice_str == '1':
        # Perform Addition
        print('Enter the numbers you would like to add \n')
        flt_input1 = IO.input_values_for_calculation() # Get user input
        flt_input2 = IO.input_values_for_calculation()  # Get user input
        str_result = str(Processor.perform_addition(flt_input1, flt_input2)) # Add the numbers
        IO.print_results(str_result)
    elif choice_str == '2':
        # Perform Subtraction
        pass
    elif choice_str == '3':
        # Perform Division
        pass
    elif choice_str == '4':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop
    else:
        print(choice_str + ' is not a number between 1-4')
# Work with Pickling
# Put the two together
```

After confirming the functions worked as intended I then went back through them to add a few error handling functions. An example of this can be seen below in figure 3.

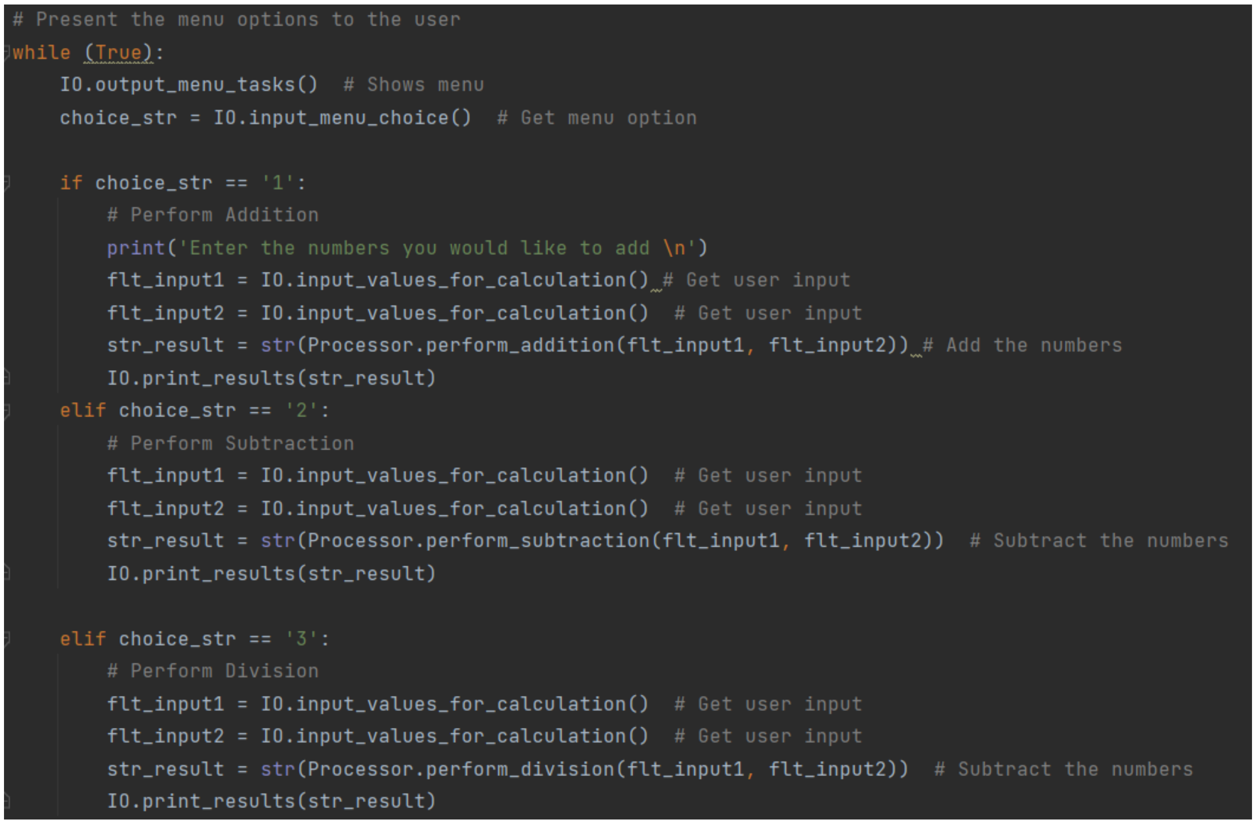
Figure 3: User Input, Value Type Error Handling



```
@staticmethod
def input_values_for_calculation():
    """ Gets a number from the user

    :return: float
    """
    try:
        flt_input = float(input('Enter a Number: '))
    except ValueError:
        print('That was not a number')
    return flt_input
```

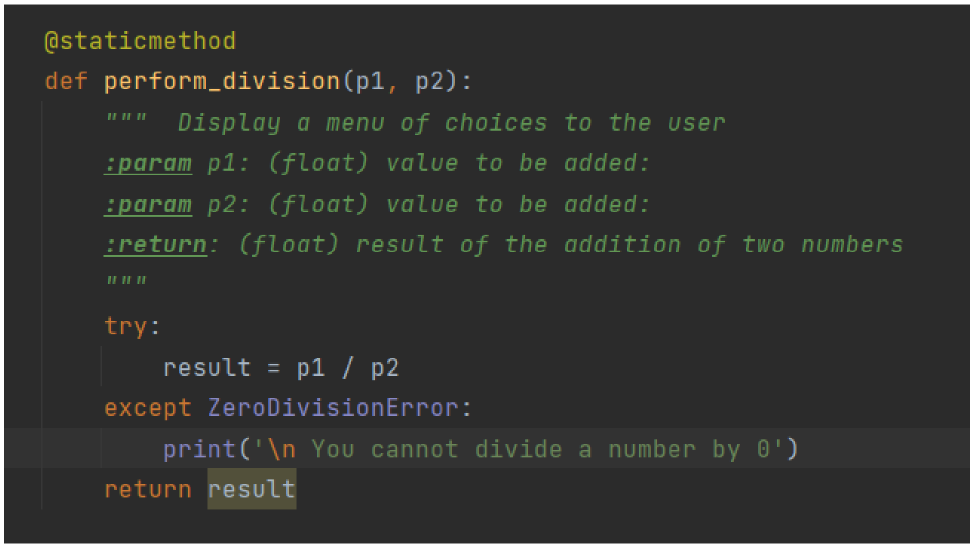
I then continued with the program by adding additional IO and processing functions to build out the subtraction and division portions of the calculator. This allowed me to try out additional error handling messages in the form of ZeroDivisionErrors shown in figure 5.

Figure 4: Additional math Operations



```
# Present the menu options to the user
while (True):
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    if choice_str == '1':
        # Perform Addition
        print('Enter the numbers you would like to add \n')
        flt_input1 = IO.input_values_for_calculation() # Get user input
        flt_input2 = IO.input_values_for_calculation()  # Get user input
        str_result = str(Processor.perform_addition(flt_input1, flt_input2)) # Add the numbers
        IO.print_results(str_result)
    elif choice_str == '2':
        # Perform Subtraction
        flt_input1 = IO.input_values_for_calculation()  # Get user input
        flt_input2 = IO.input_values_for_calculation()  # Get user input
        str_result = str(Processor.perform_subtraction(flt_input1, flt_input2))  # Subtract the numbers
        IO.print_results(str_result)

    elif choice_str == '3':
        # Perform Division
        flt_input1 = IO.input_values_for_calculation()  # Get user input
        flt_input2 = IO.input_values_for_calculation()  # Get user input
        str_result = str(Processor.perform_division(flt_input1, flt_input2))  # Subtract the numbers
        IO.print_results(str_result)
```

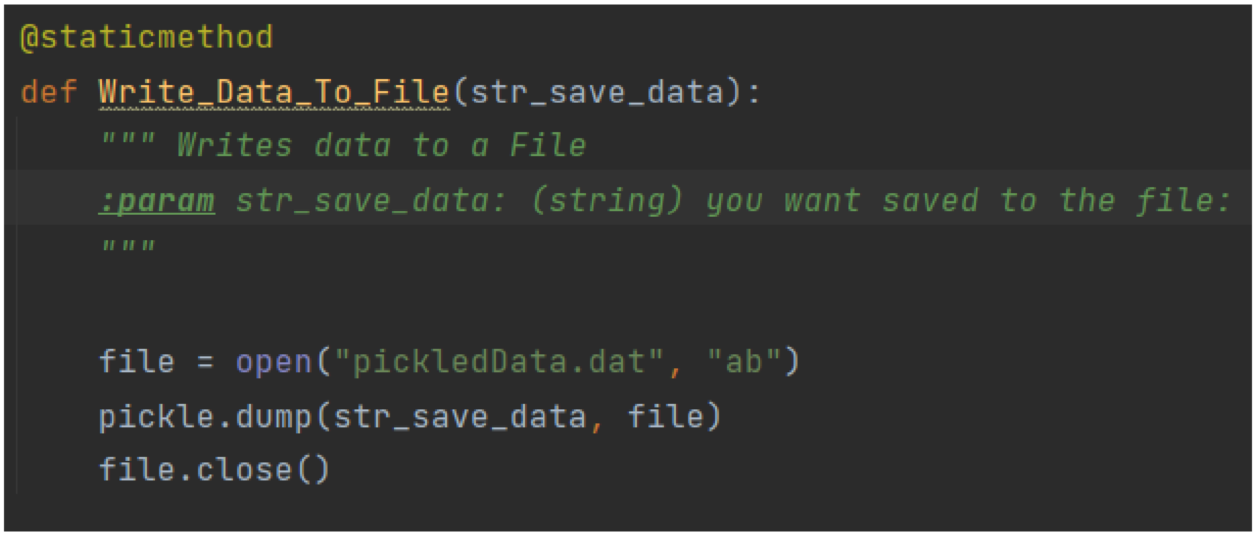
Figure 5: Try/Except ZeroDivisionError



```
@staticmethod
def perform_division(p1, p2):
    """  Display a menu of choices to the user
    :param p1: (float) value to be added:
    :param p2: (float) value to be added:
    :return: (float) result of the addition of two numbers
    """
    try:
        result = p1 / p2
    except ZeroDivisionError:
        print('\n You cannot divide a number by 0')
    return result
```

The next thing I did was work set about implementing pickling in the program. To do this I wrote a function to ask the user if they would like to save the data and another to do the saving. The save data to file function is listed below. I was fairly time constrained this week on business travel. Given more time, I would like to have taken the file name out of the function and moved it into a variable.

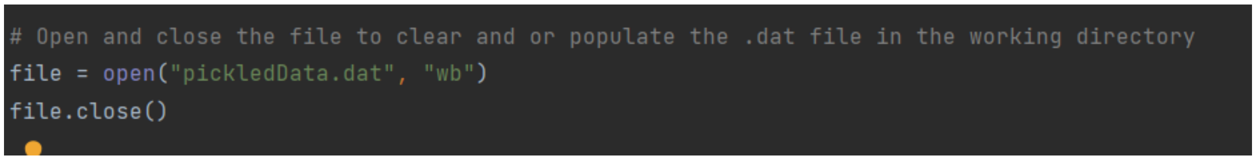
Figure 6: Write data to a file



```
@staticmethod
def Write_Data_To_File(str_save_data):
    """ Writes data to a File
    :param str_save_data: (string) you want saved to the file:
    """

    file = open("pickledData.dat", "ab")
    pickle.dump(str_save_data, file)
    file.close()
```

After creating these functions I then added lines to the addition, subtraction, and division portions of my main program to give the user the option to save the formulas. I then added a few lines at the beginning of the main to open and clear the file at the start of each new sessions. This was partially to ensure the file was created in the working directory as well as to ensure only data from the current session was being saved.

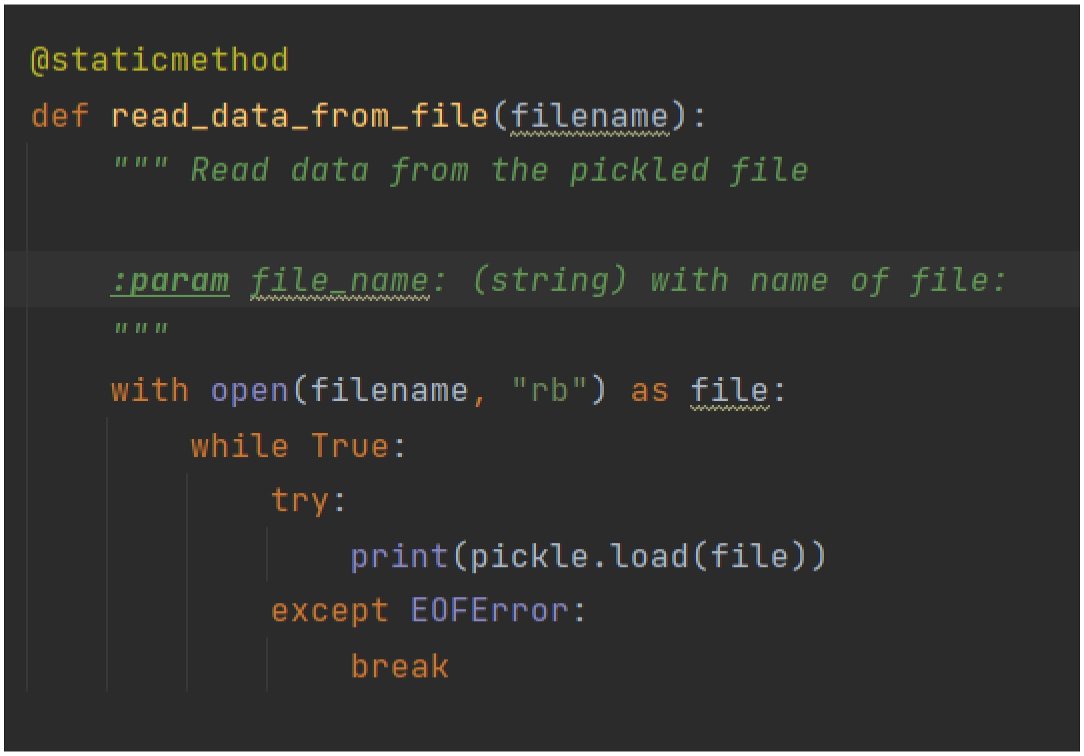
Figure 7: Pickle File Generation



```
# Open and close the file to clear and or populate the .dat file in the working directory
file = open("pickledData.dat", "wb")
file.close()
```

From here, I tested the program, worked through a few bugs and ensured everything worked as expected. I then set out to add a user option to review data in the file. This proved to be the most difficult portion of the program and took some searching on the internet to help create a solution. After messing around with for loops to try and loop through lines of the dat file while loading data unsuccessfully, I came upon an elegant solution that utilized try/except error handling. How fortuitous. The website I got the help from was How to save and load multiple objects in pickle file with Python? - The Web Dev. After modifying the code to fit my specific situation, i arrived at a processing function shown in figure 8.

Figure 8: Read data from file

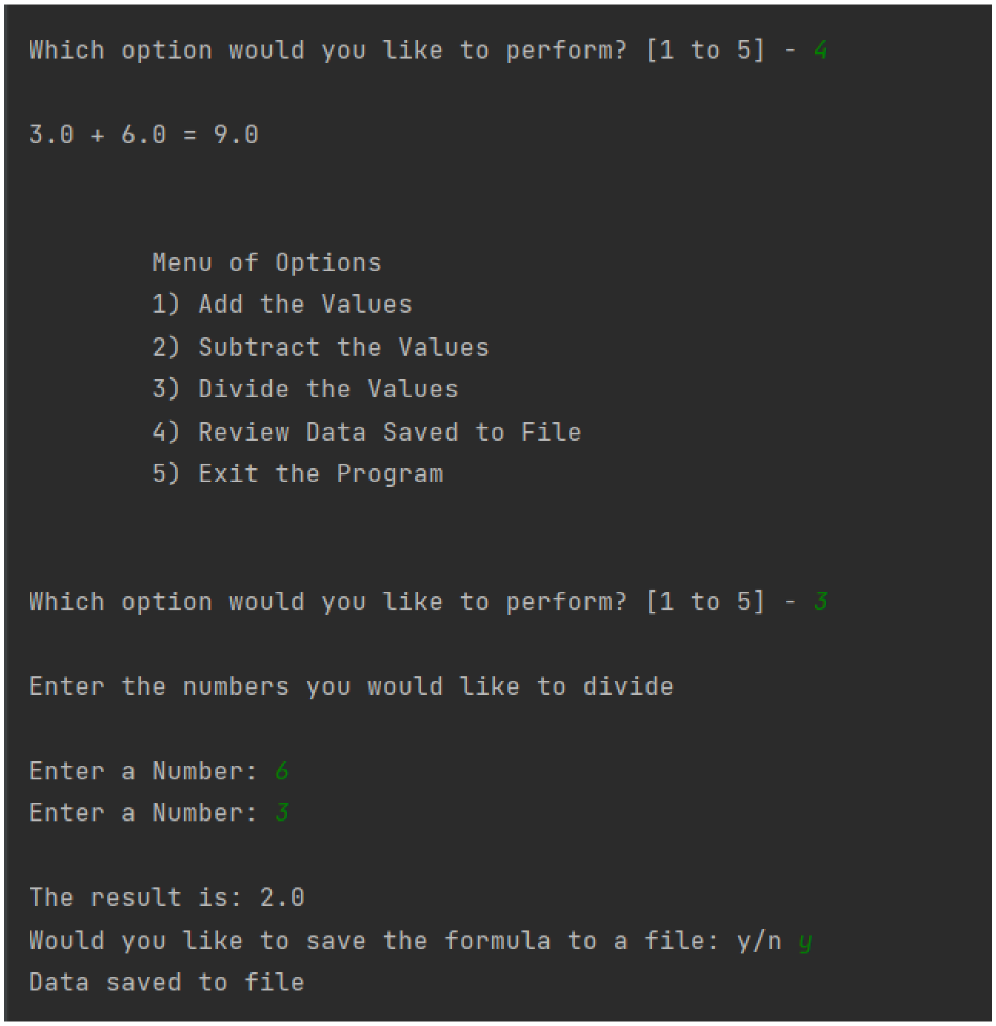


```
@staticmethod
def read_data_from_file(filename):
    """ Read data from the pickled file

    :param file_name: (string) with name of file:
    """
    with open(filename, "rb") as file:
        while True:
            try:
                print(pickle.load(file))
            except EOFError:
                break
```

From here, my program was pretty much complete. I added the calls to the functions to the main body of the program and ran through a few bug fixes before calling the program complete.

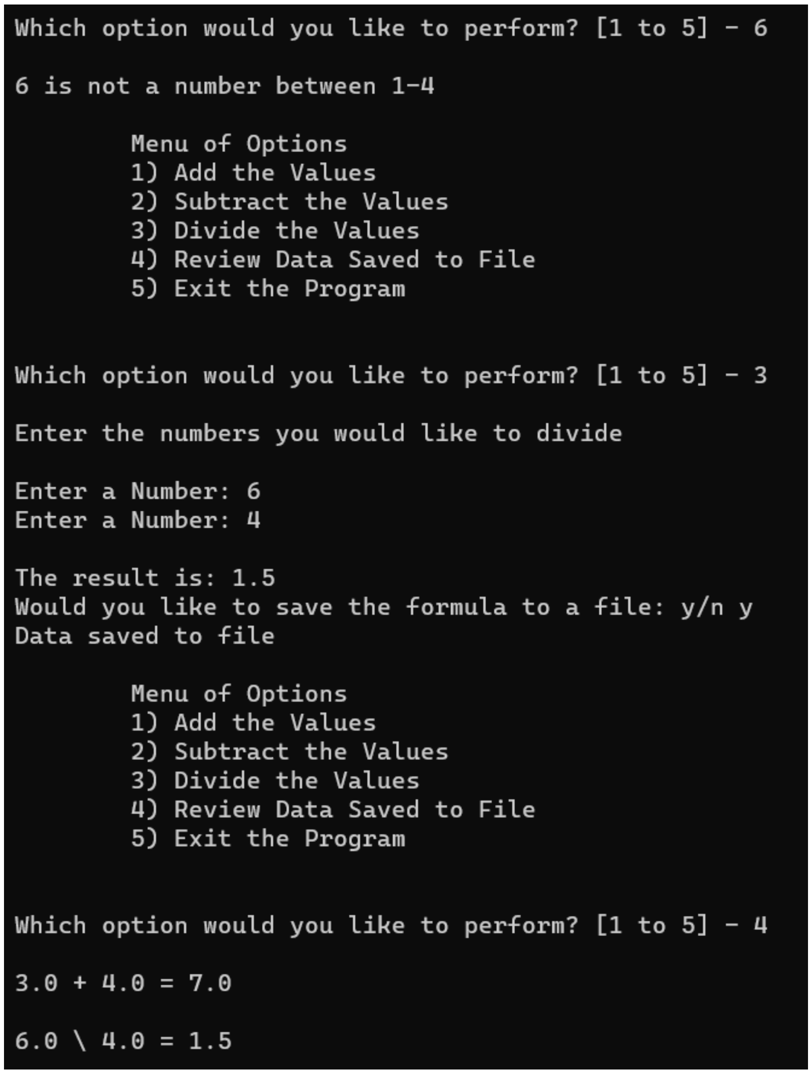
Conclusion:
In this module, I reviewed the course material and leveraged external sites to learn about pickling and error handling. I then implemented a bit of code to demonstrate these topics. The execution of these files can be seen in figures 9 and 10 below.   The files are saved on Github here: zdevries123/Intro_to_Python_Module07 (github.com)

Figure 9: Program Execution in PyCharm



```
Which option would you like to perform? [1 to 5] - 4

3.0 + 6.0 = 9.0



        Menu of Options
        1) Add the Values
        2) Subtract the Values
        3) Divide the Values
        4) Review Data Saved to File
        5) Exit the Program



Which option would you like to perform? [1 to 5] - 3

Enter the numbers you would like to divide

Enter a Number: 6
Enter a Number: 3

The result is: 2.0
Would you like to save the formula to a file: y/n y
Data saved to file
```

Figure 10: Program Execution in Console



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

6 is not a number between 1-4

        Menu of Options
        1) Add the Values
        2) Subtract the Values
        3) Divide the Values
        4) Review Data Saved to File
        5) Exit the Program


Which option would you like to perform? [1 to 5] - 3

Enter the numbers you would like to divide

Enter a Number: 6
Enter a Number: 4

The result is: 1.5
Would you like to save the formula to a file: y/n y
Data saved to file

        Menu of Options
        1) Add the Values
        2) Subtract the Values
        3) Divide the Values
        4) Review Data Saved to File
        5) Exit the Program


Which option would you like to perform? [1 to 5] - 4

3.0 + 4.0 = 7.0

6.0 \ 4.0 = 1.5
```