Bhawna Tyagi

November 30, 2021

Foundations of Programming: Python

Assignment 07

< GitHub-link>

Structured Error Handling and Pickling

# Introduction:

This assignment is about understanding and experimenting on how to handle errors/exceptions in python code. As part of this exercise, I will walk you through one example code where I will be creating student dictionary having keys as student name and the course in which they have enrolled themselves. Within the same example, I have introduced pickling which explains how the data can be stored in binary format in python. That being said, in the upcoming sections, I will show you how I have created different classes to performed different operations and the expected exceptions that can happen while performing operations on the list of student dictionaries.

# What is an exception and exception handling in python?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program’s instructions. In general, when a python script encounters a situation that it cannot cope with, it raises an exception. An exception is a python object that represents an error. If you have some suspicious code that may raise an exception, you can defend your program by placing suspicious code in a try block. After the try block, you must include an except statement followed by a block of code which handles the exception.

# What is pickling and unpickling in python?

In python, pickle module is used to serialize or deserialize python object structures. The process of converting any python objects (i.e., list, dictionary) into byte streams is called pickling. We can also convert the byte streams back into python objects. This process is known as unpickling.

# Breakdown of python script:

# Different classes to handle Processing/IO tasks and Exception handling:

Below are the three classes that I have created in my python script:

# StudentProcessor:

This class is responsible for processing of the below tasks:

1. Reading student data from the text file
2. Adding more student dictionary objects in the student list.
3. Saving the student data to the new file.
4. Pickling of the student data.
5. Unpickling of the student data.

# IOStudentProcessor:

This class is responsible for printing out the student menu options as well as getting user input to perform certain tasks using student menu. Following are the tasks:

1. Gets the user choice from the student menu options.
2. Show the current student menu options to the user.
3. Get the user input to add new student name and course.
4. Asking user to choose from y/n to continue the process or not.

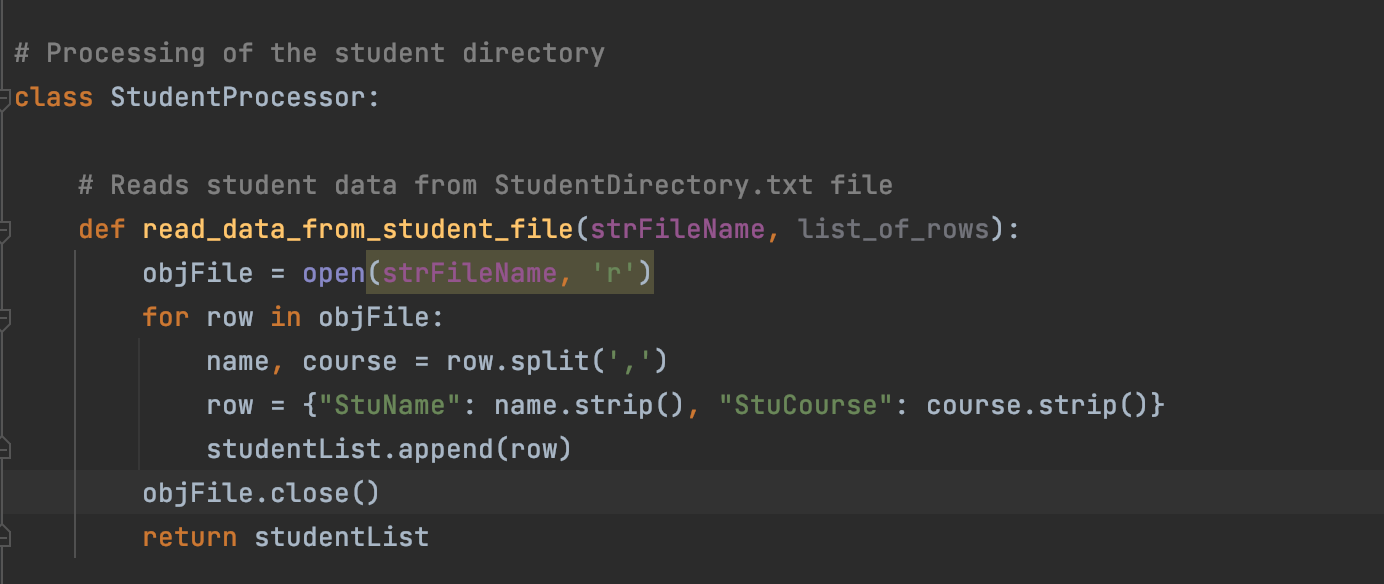
# ExceptionStudentCls:

This class is responsible for handling different exceptions such as TypeError as well as CustomError.

# Different methods within each class:

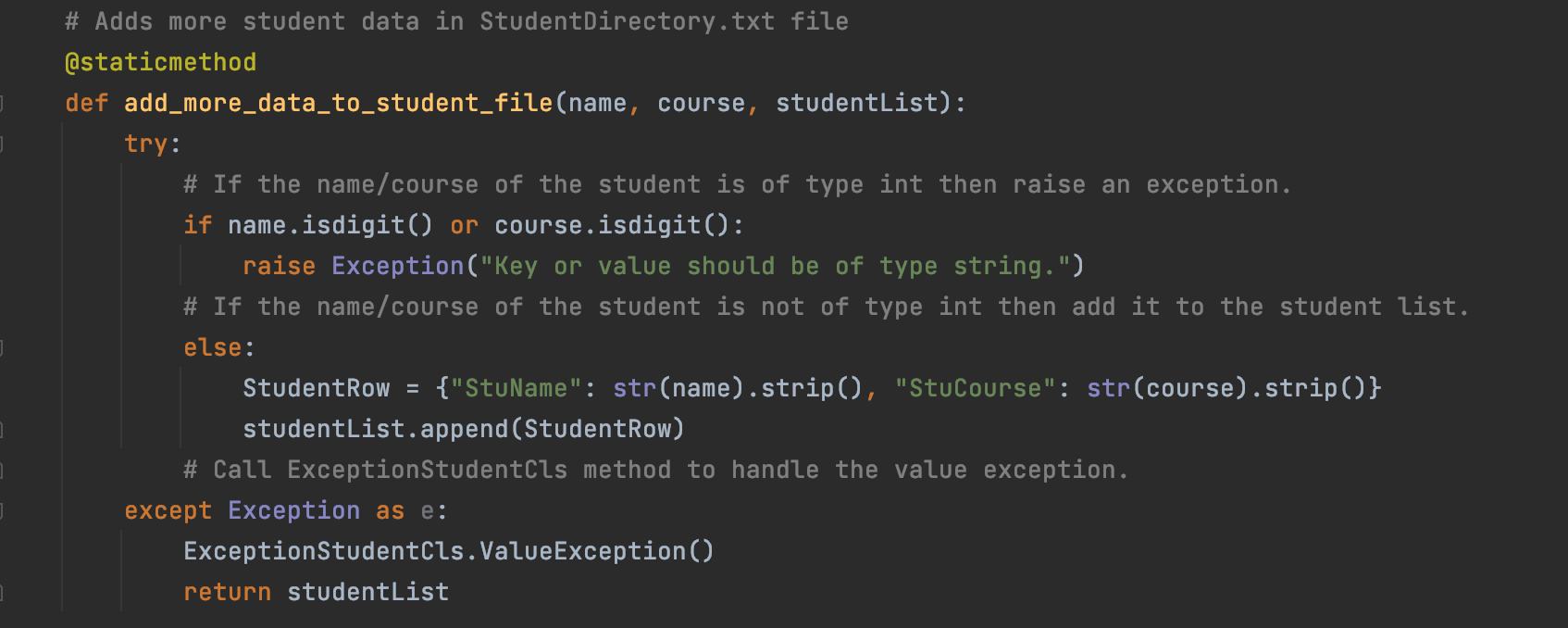
1. **StudentProcessor class methods:**

**Read\_data\_from\_file (strFileName, list\_of\_rows):** The purpose of this method is to read the file which has student data. Basically, this file contains student names and courses as rows separated by comma and create dictionary object. Finally, create list of student dictionary objects. This method takes file name that contains student data and student list. The function starts with opening the file in the read mode and then split the student row using split (). On splitting the row, we will store student name in name variable and student’s corresponding course in course variable. Using these two variables, we create row and finally add that row in the studentList. This list i.e., (studentList) has been initialized as an empty list to store the dictionary objects. After adding all student names and courses in a list, we close the file and return studentList as result. The complete method logic is shown in Figure 1.



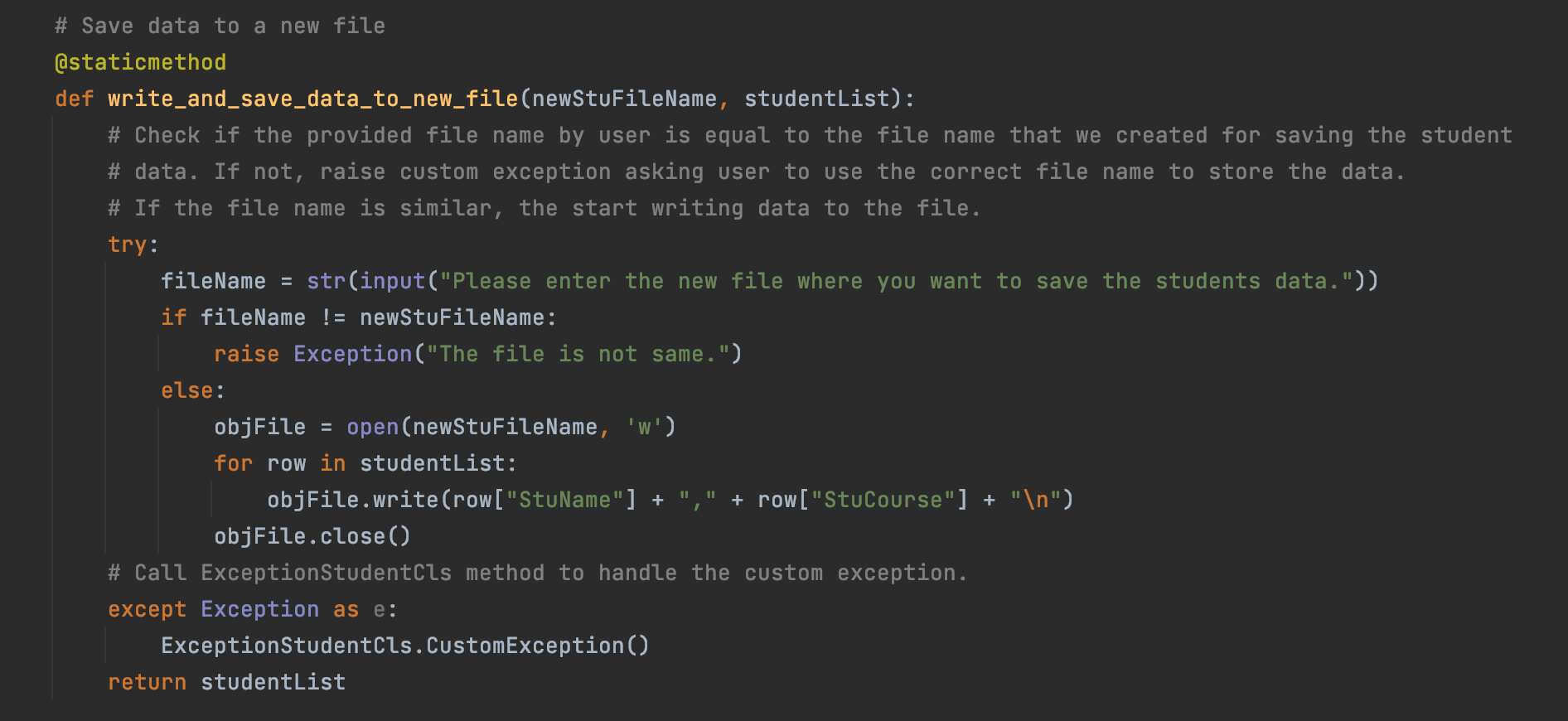
***Figure 1. Reading data from file***

**Add\_more\_data\_to\_student\_file (name, course, studentList):** The purpose of this method is to add new student name and course in the studentList after checking whether the provided student name and course are of type string. The code has been wrapped in try block. The method takes in name, course and studentList as input parameters. In the try block, the logic is to check whether the provided student name and course is of type string. If it is of type string then create row and add the row into the existing studentList. If not, the exception will be raised by calling ValueException () method of ExceptionStudentCls. There is another general exception which gets raised on checking whether the name and course is of type. Overall, two exceptions are raised together in block of code. Finally, we return studentList as result. The complete method logic is shown in Figure 2.



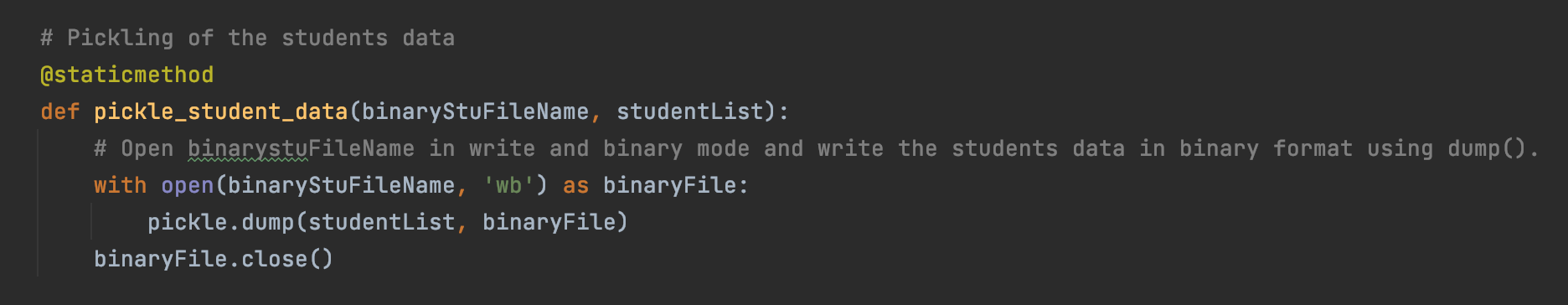
***Figure 2. Adding new student data in studentList***

**Write\_and\_save\_data\_to\_new\_file (newStuFileName, studentList):** The purpose of this method is to save the student data to a new file named as RawStudentDirectory.txt. The method starts with try block that contains logic where we first take the file name where user wants to write the data. Then, we check whether the user provided file name is similar to what has been mentioned in the code initially i.e., RawStudentDirectory.txt. If the file names are same then we open the new file in write mode and iterate through each row in the studentList and write it to the new file. In case if the file names are different, then we raise general exception with message “The file is not same”. Another exception is raised in except block which calls CustomException () of ExceptionStudentCls with message “CustomError: Please use the correct file name!”. Finally, we return studentList as a result. The complete method logic is shown in Figure 3.



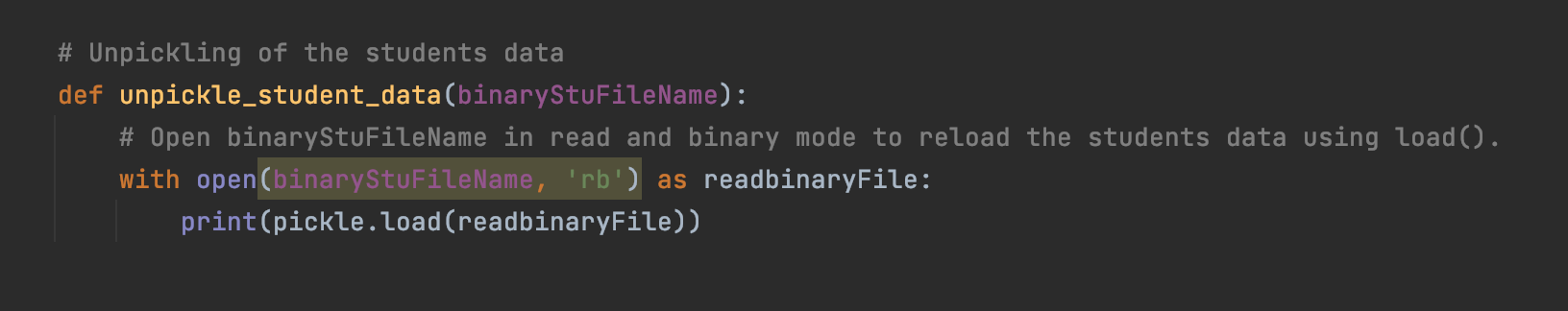
***Figure 3. Saving data in new file***

**Pickle\_student\_data (binaryStuFileName, studentList):** The purpose of this method is to convert the list object into byte streams. This method takes file name where we want to store the data in binary format along with studentList. The logic starts with opening the file in wb mode (i.e., write + binary) mode. Finally, we use dump () method which takes two parameters i.e., studentList and binaryFile which is a file object. After writing all rows of the studentList, we close the file using close (). The complete method logic is shown in Figure 4.



***Figure 4. Pickling data logic***

**Unpickle\_student\_data (binaryStuFileName):** The purpose of this method is to convert the byte streams back to list object. This method takes binary file name as an input parameter. We again open the file in read + binary mode and finally use load () to read the binary file in text format. The complete method logic is shown in Figure 5.



***Figure 5. Unpickling data logic***

**2. IOStudentProcessor class methods:**

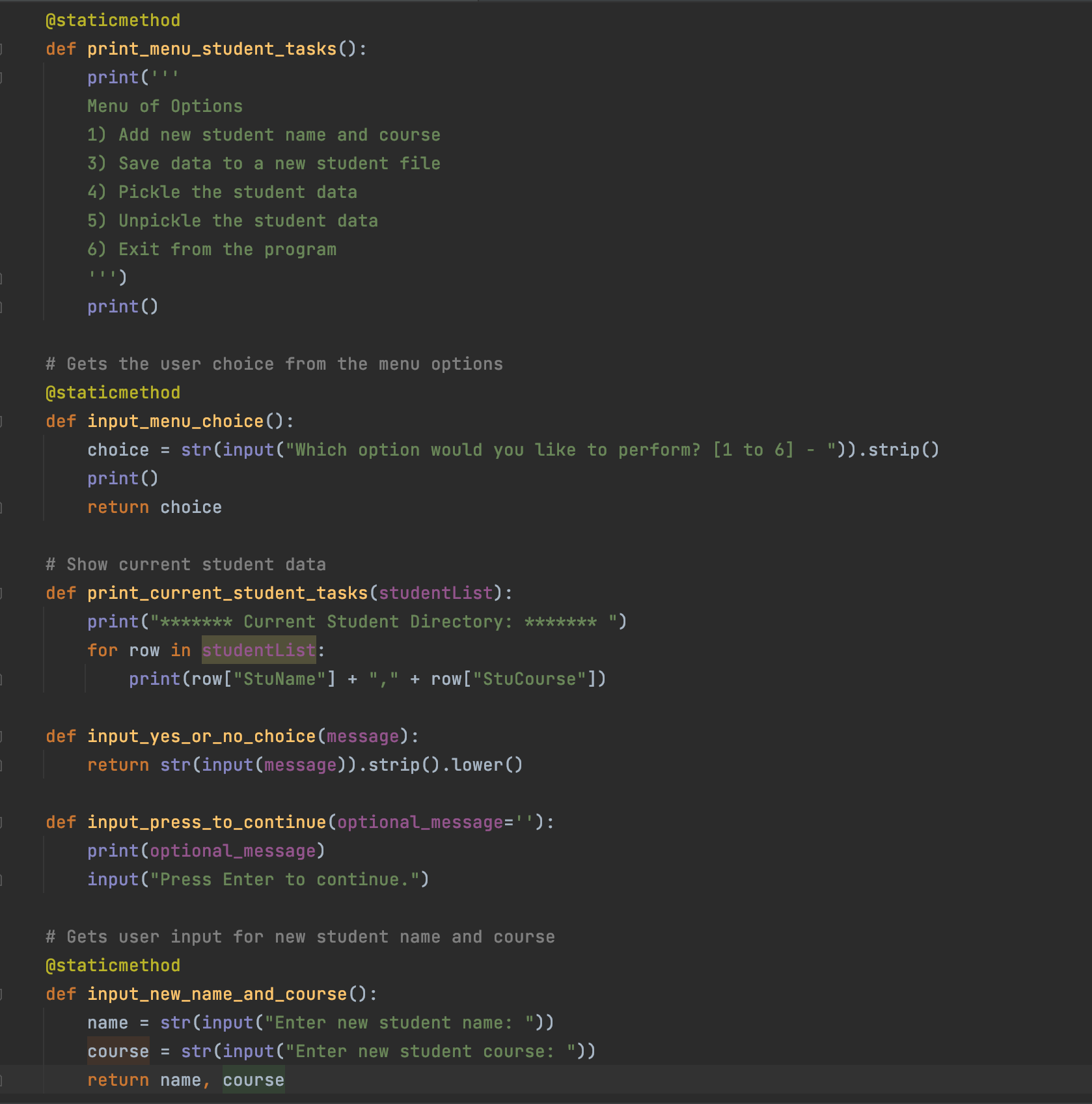
**Print\_menu\_student\_tasks ():** The purpose of this method is to print the menu options. The complete logic is shown in Figure 6.

**Input\_menu\_choice ():** The purpose of this method is to get the user choice among the provided student menu options. As a result, we return choice of the user. The complete logic is shown in Figure 6.

**Print\_current\_student\_tasks ():** The purpose of this method is to print the current student objects in the studentList. The complete logic is shown in Figure 6.

**Input\_yes\_or\_no\_choice ():** The purpose of this method is to return the message in terms of y/n when the logic asks user whether they want to continue with the instructions to finish the tasks. The complete logic is shown in Figure 6.

**Input\_new\_name\_and\_course ():** The purpose of this method is to take new student name and course from user as input and return both parameters as a result. The complete logic is shown in Figure 6.

****

***Figure 6. IOStudentProcessor class methods***

3.**ExceptionStudentCls class methods:**

**ValueException ():** The purpose of this method is to raise an exception when the provided student name and course is not of type string. The complete logic is shown in Figure 7.

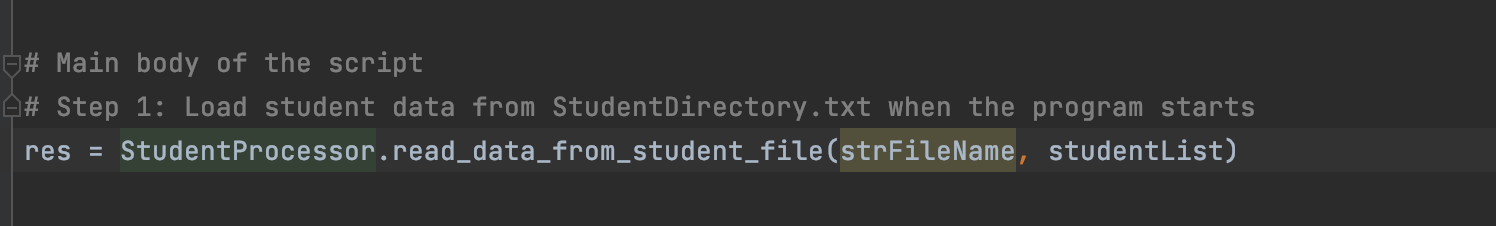
**CustomException ():** The purpose of this method is to raise custom exception when the provided file name to save the student data is not similar to the one mentioned in the python script. The complete logic is shown in Figure 7.

# 

***Figure 7. ExceptionStudentCls class methods***

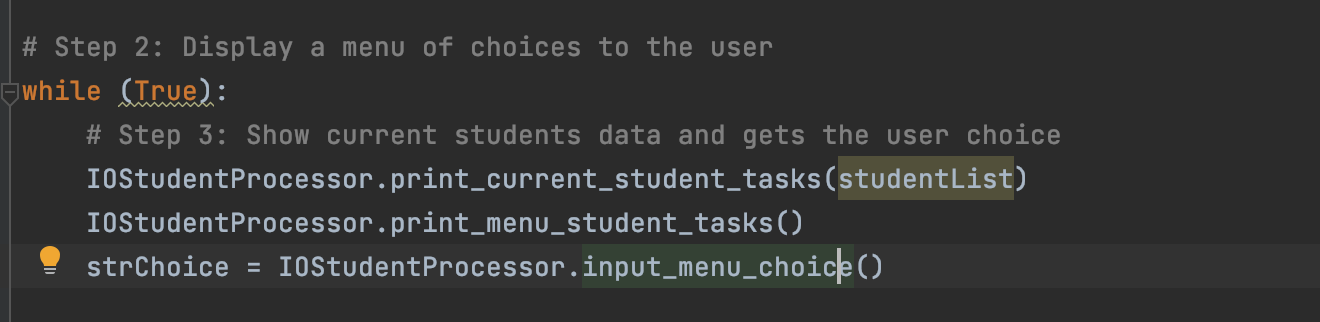
# Main body of the script:

Step 1: The program starts with calling the read\_data\_from\_student\_file () in order to load the data from the file as shown in Figure 8.



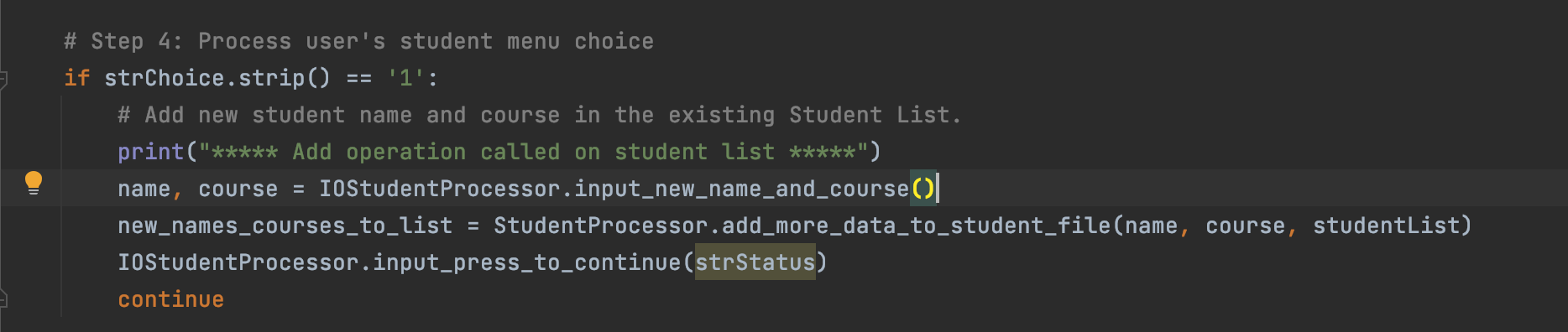
***Figure 8. Beginning of the Main script of the body by loading the data from the file***

**Step 2:** Next step is to show the menu options to the user along with getting the user choice. This is done by calling the IOStudentProcessor class methods as shown in Figure 9.

******

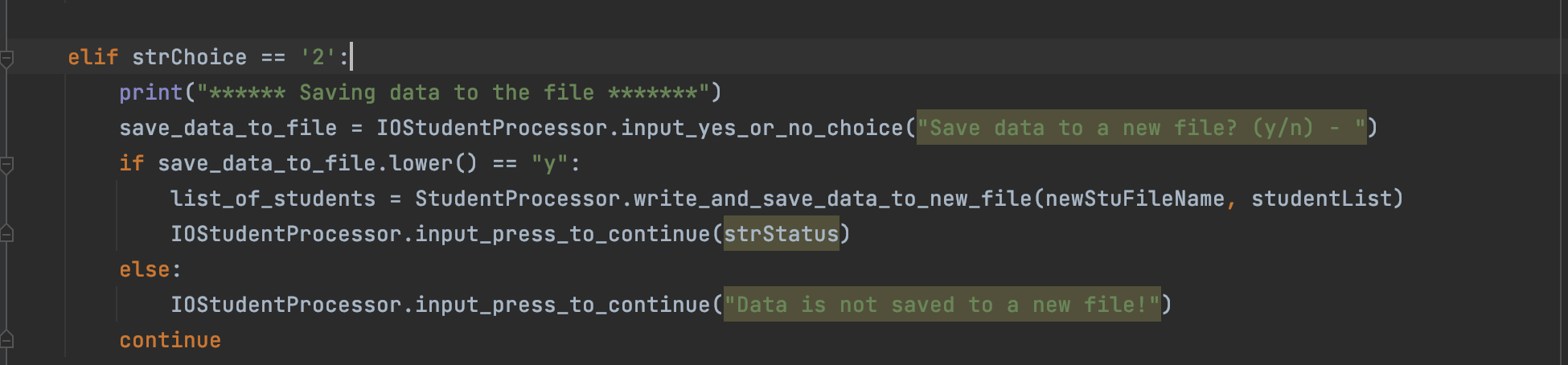
***Figure 9. Calling IOStudentProcessor Class methods***

**Step 3:** If the user chooses option 1, this means that user wants to add new student names and enrolled courses in the existing student list. In order to achieve this, first we ask user to provide the student ‘s name and course that needs to be added to the list and finally we call the methods that would do this processing on behalf of the user. Finally, we ask user whether they want to continue with the next action or not. The complete logic is shown in Figure 10.

******

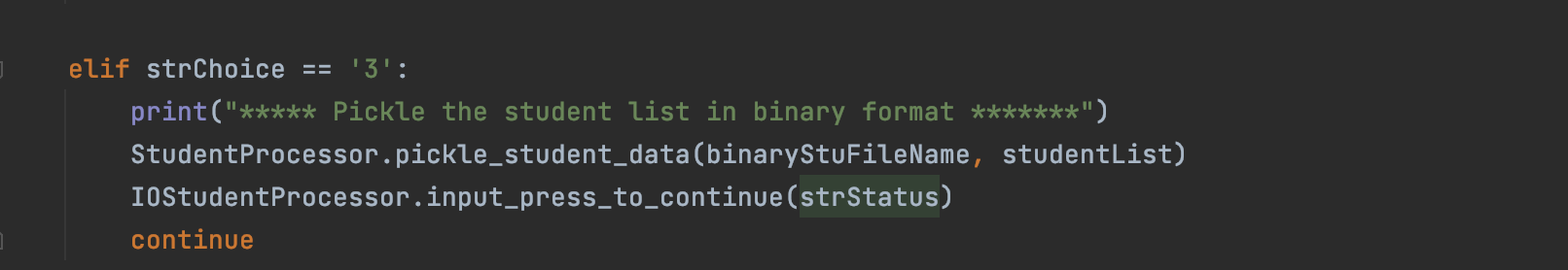
***Figure 10. Logic for student menu option 1***

**Step 4:** If user chooses option 2, this means that user wants to save the data to the new file. In order to achieve this, first we ask user whether they want to save the data to the file or not. If the input is ‘y’ then the method would be called which will be responsible for saving the data to the file. Finally, we ask user whether they want to continue with the next action or not. In case, if they don’t want to proceed with saving data to a file, then print the message to the user “Save canceled”. This means that data will not be saved to the file. The complete logic is shown in Figure 11.



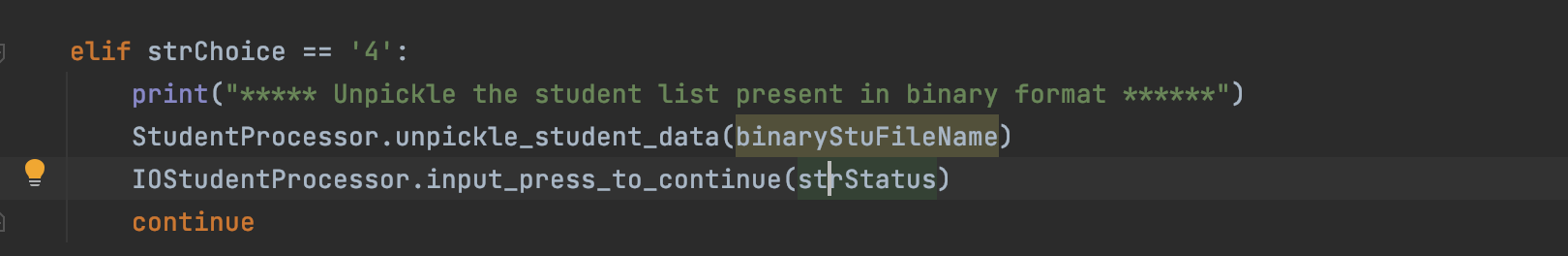
***Figure 11. Logic for student menu option 2***

**Step 5:** If user chooses option 3, this means that user wants to convert the data into binary format. If this is the case then we would call pickle’s dump method to achieve this task. The complete logic is shown in Figure 12.



***Figure 12. Logic for student menu option 3***

**Step 6:** If user chooses option 4, this means that user wants to convert the data back into list object from binary format. If this is the case then we would call pickle’s load method to achieve this task. The complete logic is shown in Figure 13.

**

***Figure 12. Logic for student menu option 4***

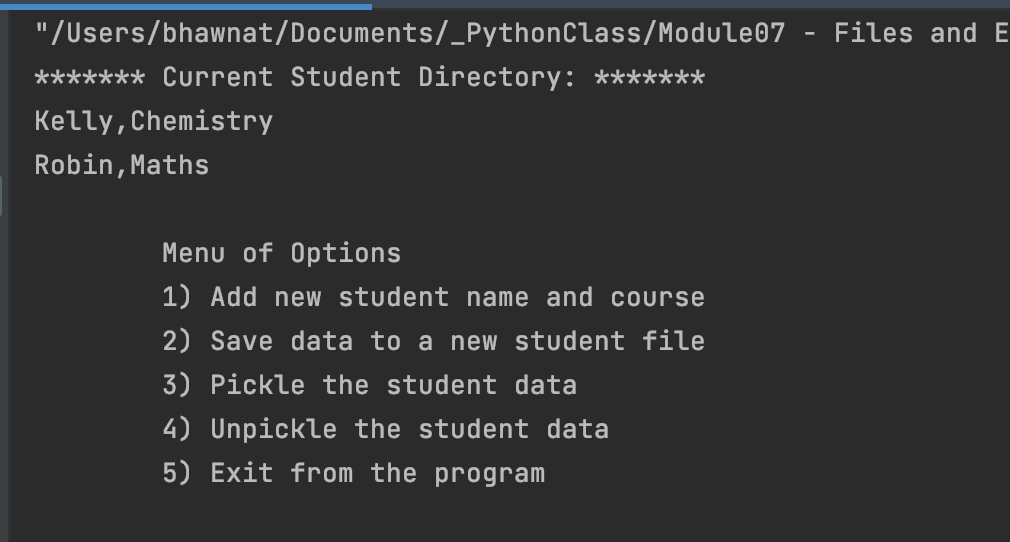
**Step 7**: If the user chooses option 7, this means that user wants to exit from the program. The complete logic is shown in Figure 14.



***Figure 13. Logic for student menu option 5***

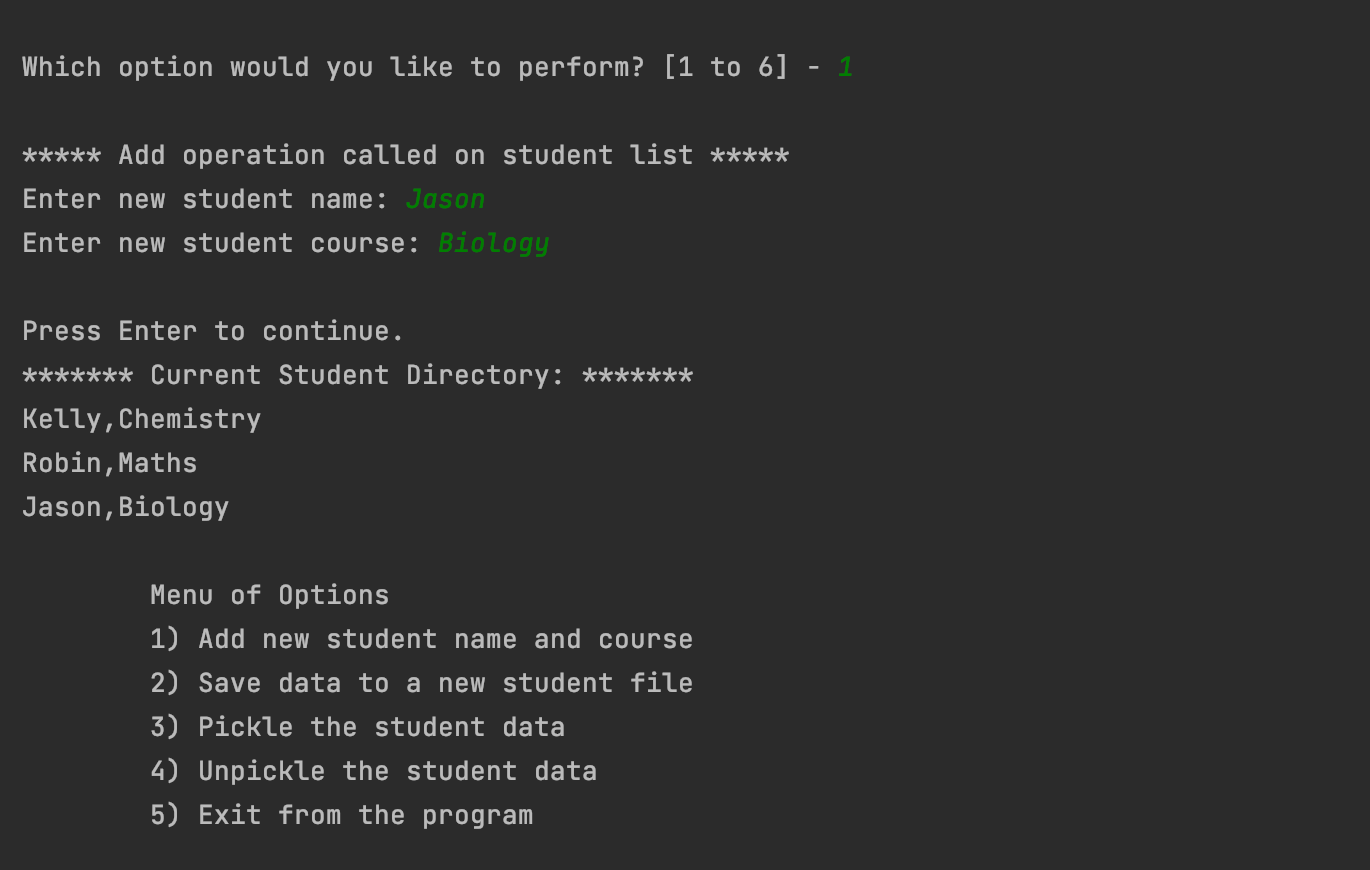
# Execution result in PyCharm:

**Scenario 1: Load data while reading the file:**



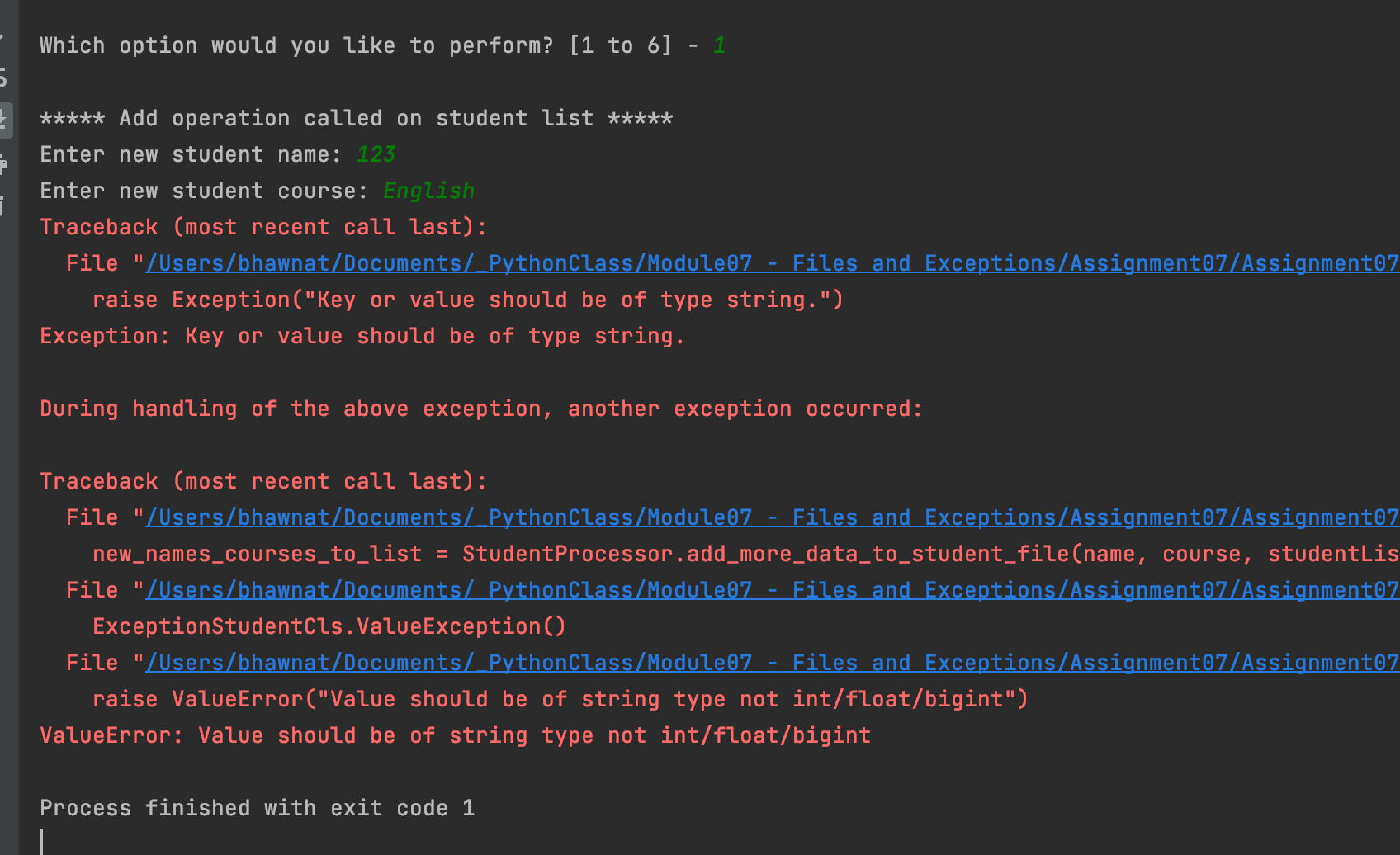
***Scenario 1***

**Scenario 2a: Successful addition of new student and course in the student list:**



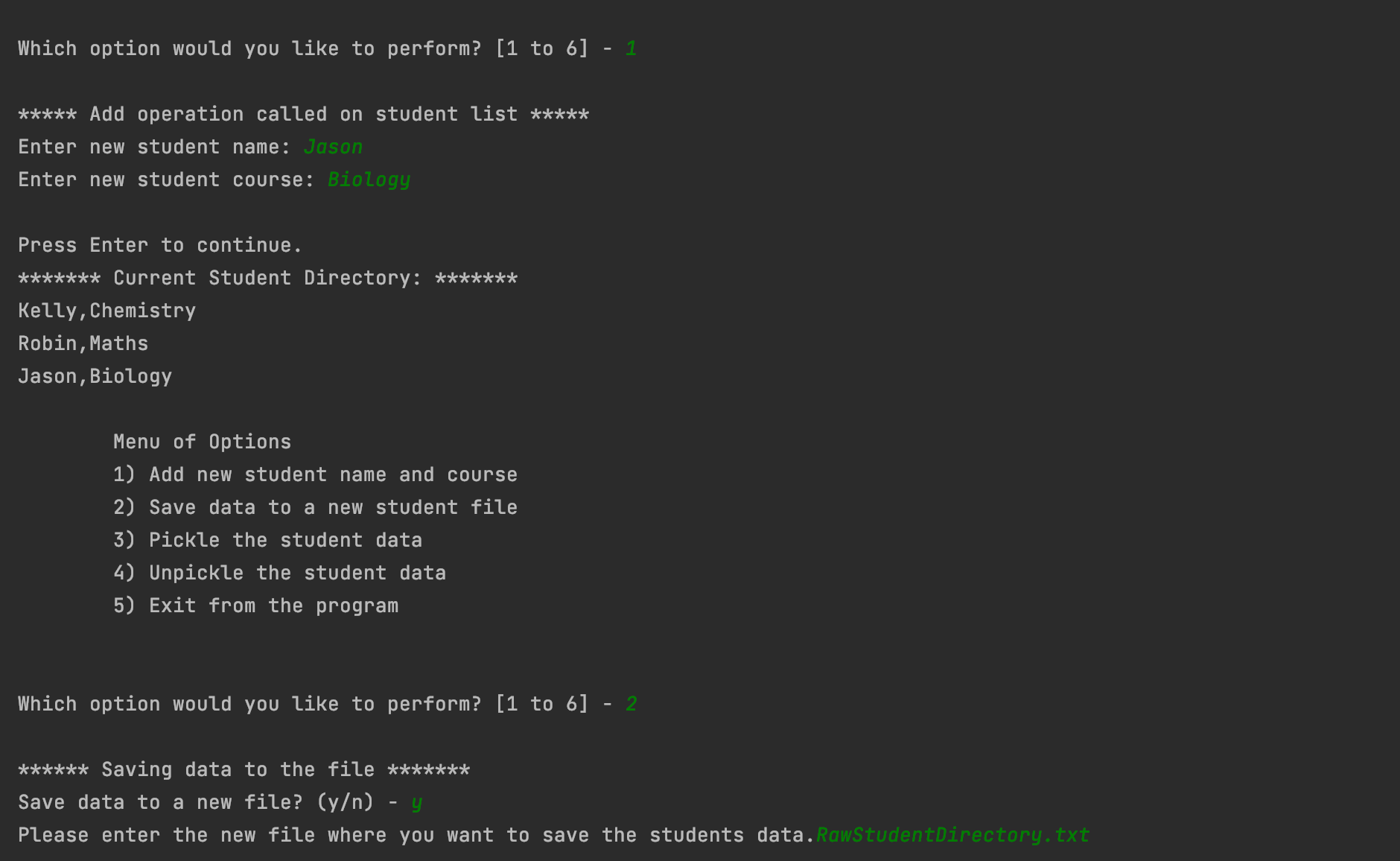
***Scenario 2a***

**Scenario 2b. Exception messages when user wants to add student name or course other than string type:**



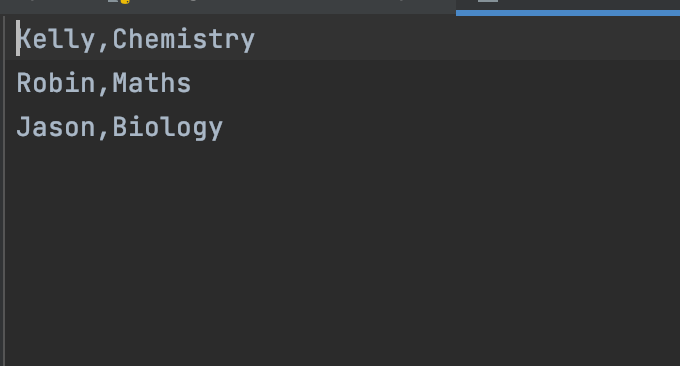
***Scenario 2b***

**Scenario 3a. Successful saving data in a file:**



***Scenario 3a***

**RawStudentDirectory.txt content:**

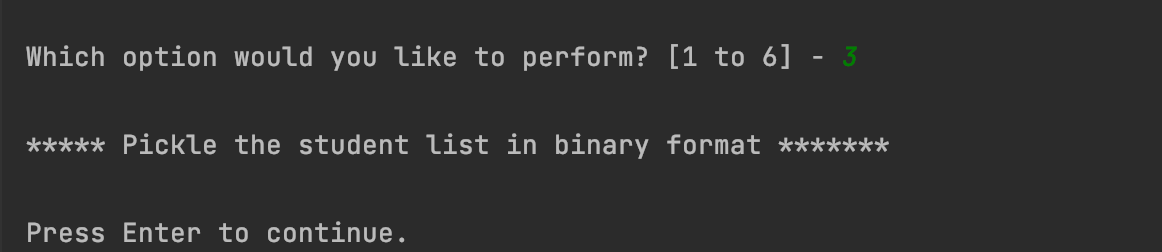


**Scenario 3b. Exception messages when user provided file name is not similar to initialized in the script with name ‘RawStudentDirectory.txt’.**

# 

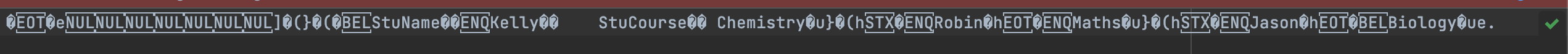
***Scenario 3b***

**Scenario 4. Pickling of the data:**

****

***Scenario 4***

**BinaryStuDir.dat file content:**

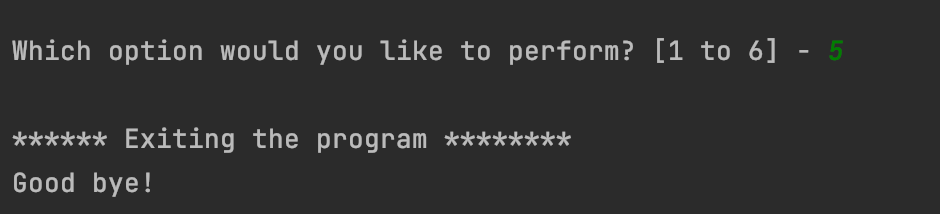
****

**Scenario 5. Unpickling of the data:**

# 

***Scenario 5***

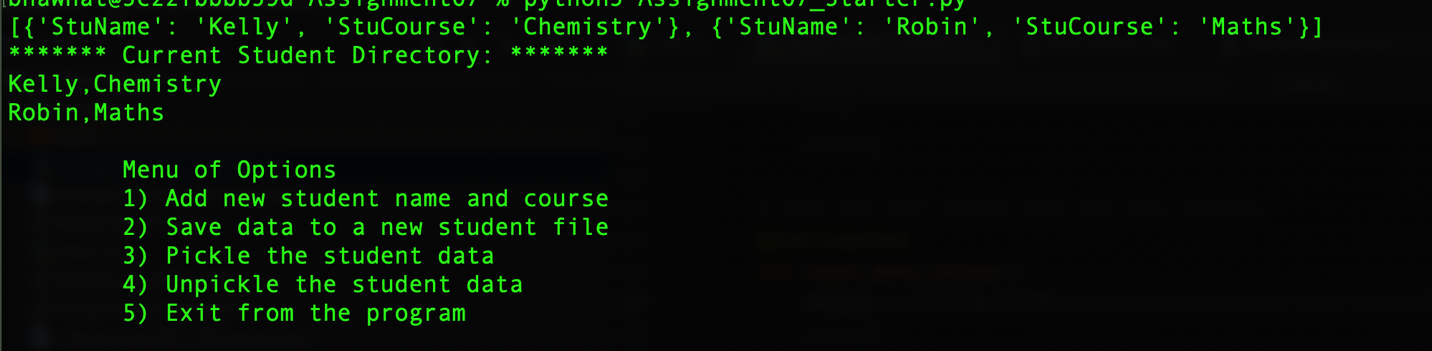
**Scenario 6. Exiting from the program:**

****

***Scenario 6***

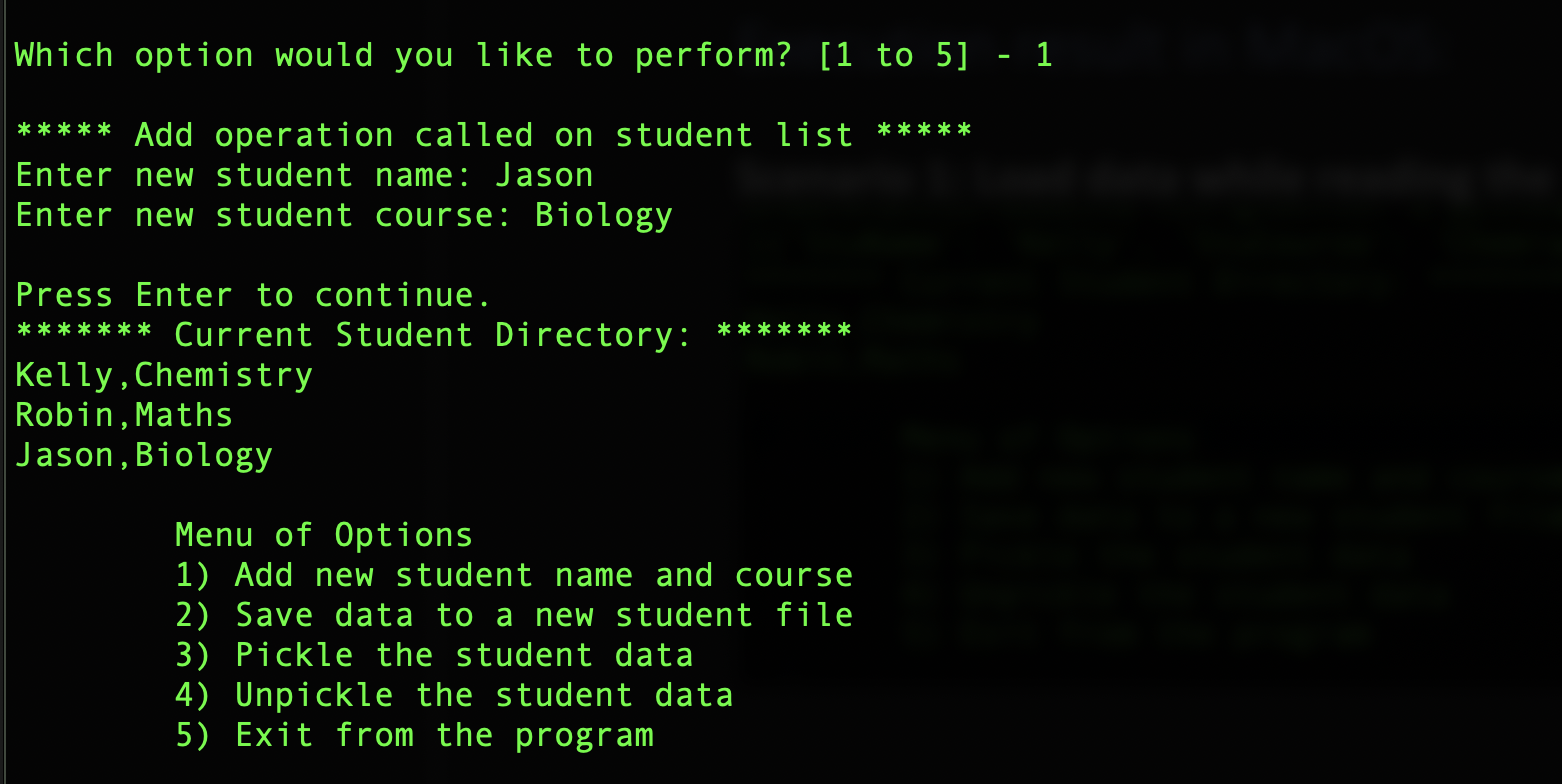
# Execution result in MacOS:

**Scenario 1: Load data while reading the file:**

****

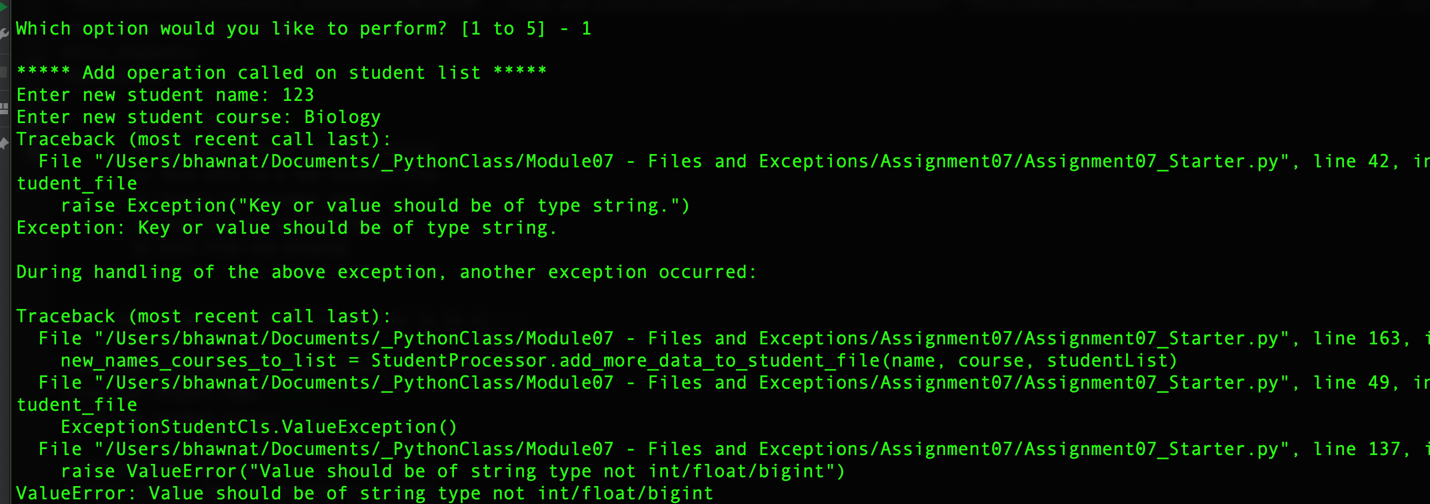
***Scenario 1***

**Scenario 2a: Successful addition of new student and course in the student list:**

****

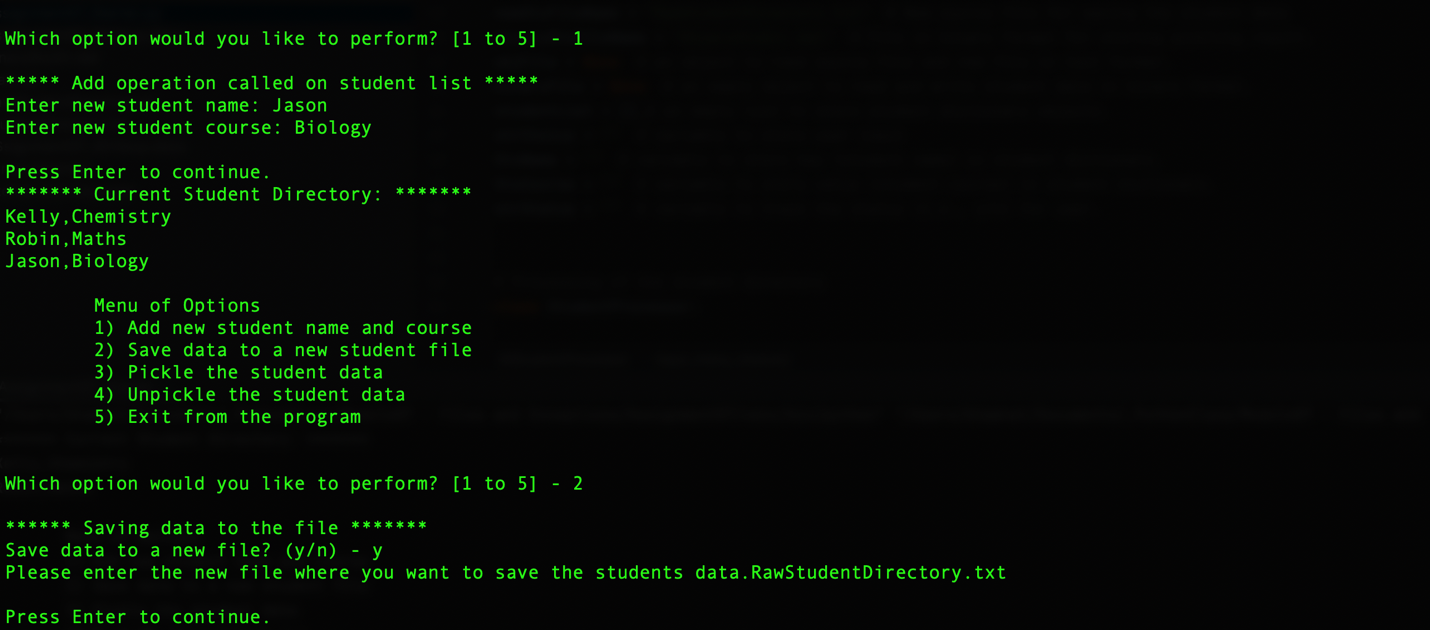
***Scenario 2a***

**Scenario 2b. Exception messages when user wants to add student name or course other than string type:**

****

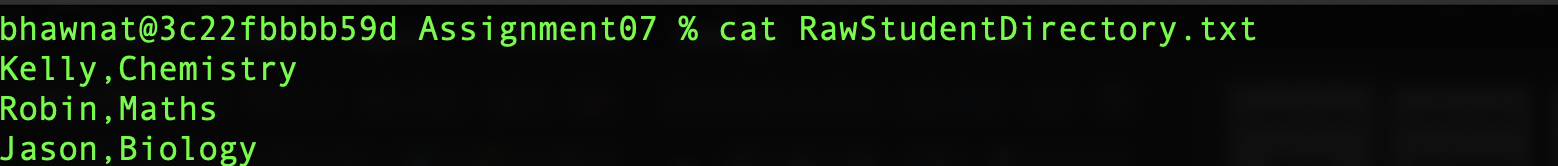
***Scenario 2b***

**Scenario 3a. Successful saving data in a file:**

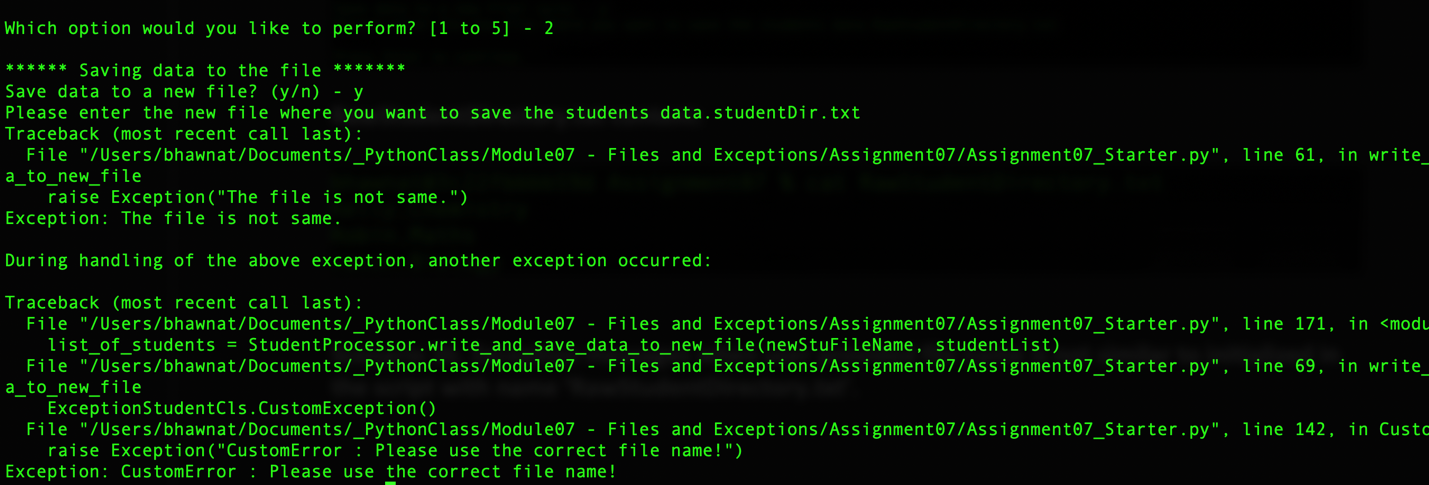
****

***Scenario 3a***

**RawStudentDirectory.txt content:**

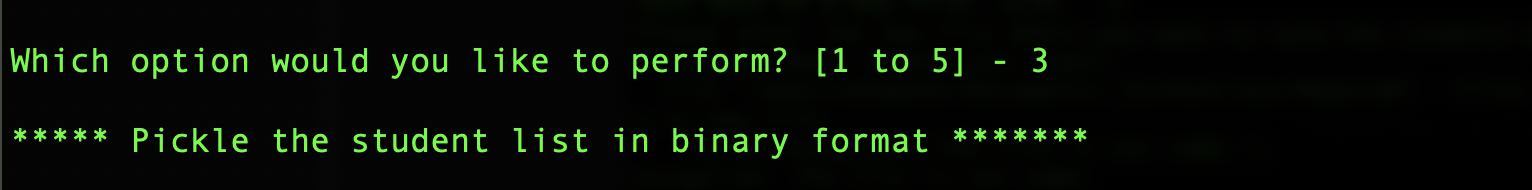
****

**Scenario 3b. Exception messages when user provided file name is not similar to initialized in the script with name ‘RawStudentDirectory.txt’.**

****

***Scenario 3b***

**Scenario 4. Pickling of the data:**

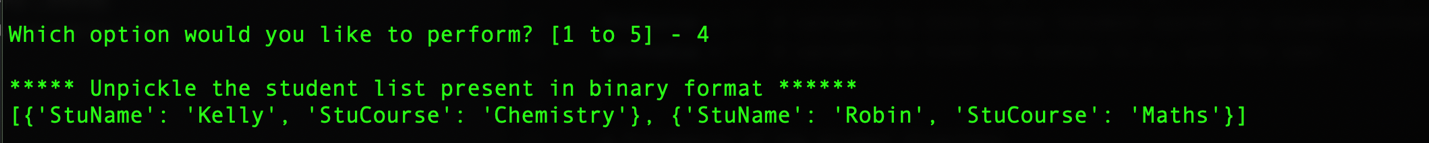
****

***Scenario 4***

**BinaryStuDir.dat file content:**

****

**Scenario 5. Unpickling of the data:**

****

***Scenario 5***

**Scenario 6. Exiting from the program:**

****

***Scenario 6***

# Summary:

In this assignment, I learned about how to handle exceptions and convert python collections into binary format using pickle module. Also, I have learned about how to deserializing the byte streams back to the python objects such as list, dictionary etc.