Jae Barnett

02/25/2025

Foundations of Programming: Python

Assignment 05

https://github.com/jaeuw

Advanced Collections & Error Handling

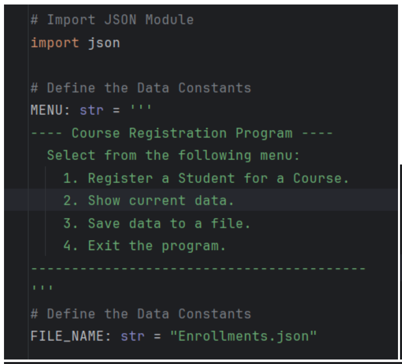
Introduction

In this document I will be creating a program that uses constants, variables and print statements to display a message about a student's registration for a Python course. I will also be incorporating the use of data processing using dictionaries and exception handling.

Declarations in Python

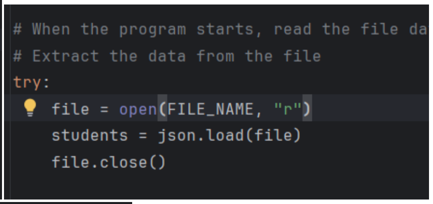
JSON

JSON stands for JavaScript Object Notation and is widely used for data serialization, configuration files and data storage. JSON files are similar to Python dictionaries since they consist of key-value pairs. The values can be strings, numbers, objects, arrays, Booleans or null with curly brackets {} for objects and square brackets [] for arrays. The keys are strings enclosed in double quotes with commas separating key-value pairs or elements within an array. For this assignment, we needed to modify the program made in the last lab to a json file. First, we needed to add an import statement to import code from the json module, change the file name to use the json extension instead of csv and add code that opens the json file, dumps the data into the students table variable and closes the file again. This is shown in the below figures 1 & 2 & 3.

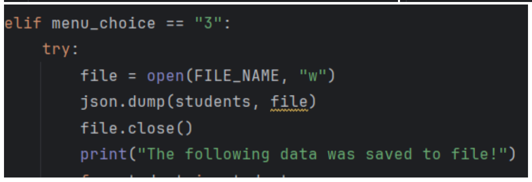


```
# Import JSON Module
import json

# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
```



```
# When the program starts, read the file da
# Extract the data from the file
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
```

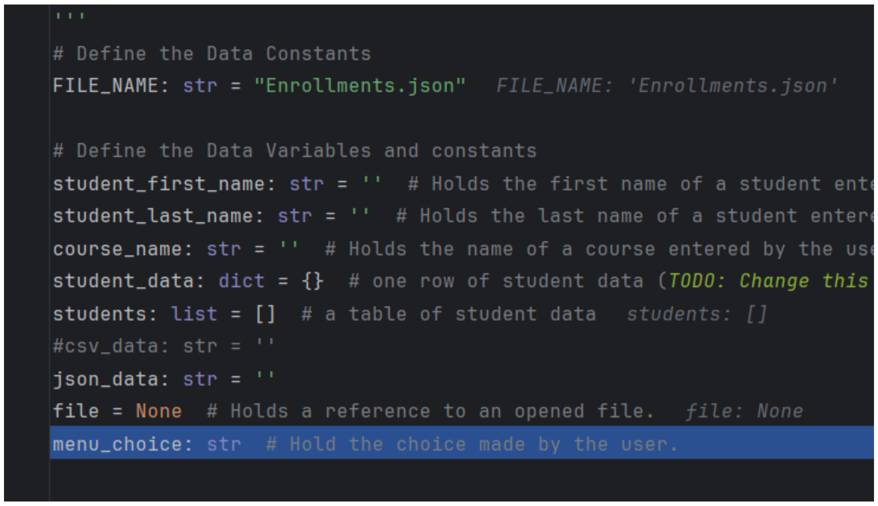


```
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")
        json.dump(students, file)
        file.close()
        print("The following data was saved to file!")
```

Figure 1 & 2 & 3: Modifications to work with json file rather than csv

Constants and Variables (including programming menu)

This program will include most of the constants and variables from assignment 4, but with some modifications. We will be changing the file_name constant to a json file, changing the student_data variable to an empty dictionary (since dict uses curly braces I changed that as well). Since we're not using a csv file I removed the csv_data variable and replaced it with a variable that works with json. With those modifications, the declarations will look like this in PyCharm:



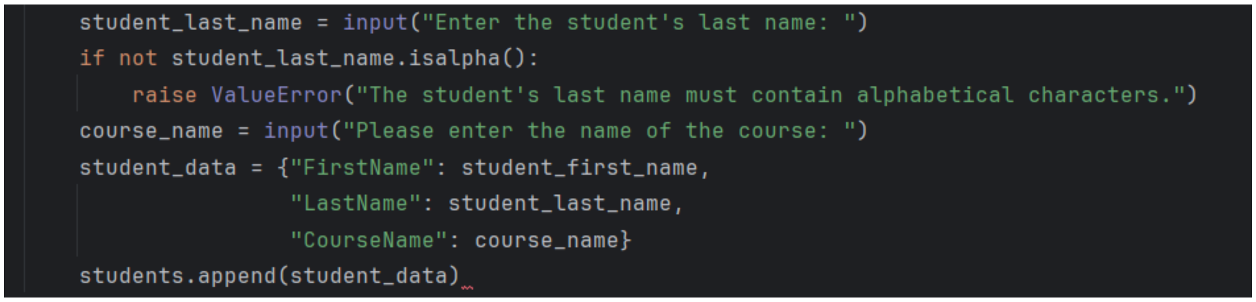
```
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"   FILE_NAME: 'Enrollments.json'

# Define the Data Variables and constants
student_first_name: str = ''  # Holds the first name of a student ente
student_last_name: str = ''  # Holds the last name of a student entere
course_name: str = ''  # Holds the name of a course entered by the use
student_data: dict = {}  # one row of student data (TODO: Change this
students: list = []  # a table of student data   students: []
#csv_data: str = ''
json_data: str = ''
file = None  # Holds a reference to an opened file.   file: None
menu_choice: str  # Hold the choice made by the user.
```

Figure 4: Defined constants and variables within the code

Dictionaries

In Python dictionaries are similar to lists in the way they store and present data, however there are a few differences. For dictionaries, in addition to the data that you put in the row you also have to include column names called "keys" that are used to identify a row with a column name. Dictionaries are enclosed within curly brackets {} and uses special built-in methods that help you work with the data, often used are items(), values(), and keys(). Keys are case-sensitive and when working with JSON uses double-quotes around the key's name. For example, the below will create a table with the columns named FirstName, LastName and CourseName.



```
student_last_name = input("Enter the student's last name: ")
if not student_last_name.isalpha():
    raise ValueError("The student's last name must contain alphabetical characters.")
course_name = input("Please enter the name of the course: ")
student_data = {"FirstName": student_first_name,
                "LastName": student_last_name,
                "CourseName": course_name}
students.append(student_data)
```

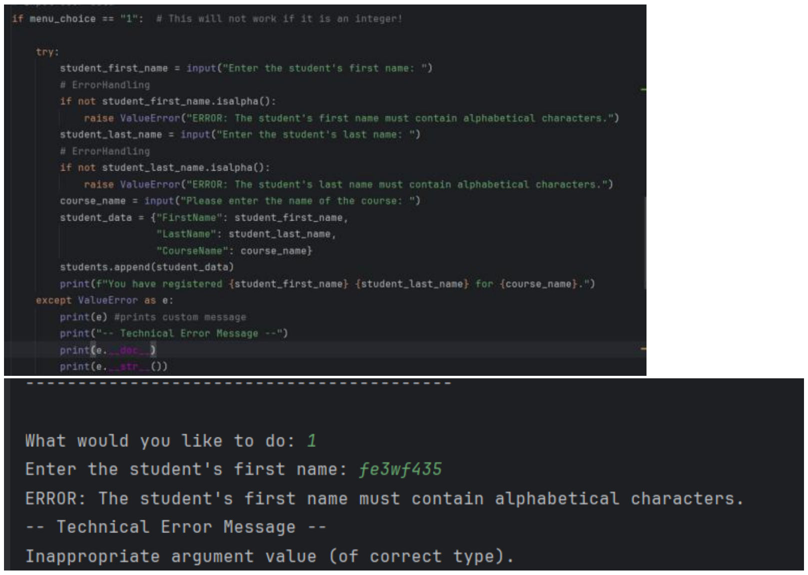
Figure 5: Defining dictionary keys and values

Error Handling

This assignment includes the use of structured error handling. When writing a program, there is always the possibility that other people introduce outside bugs when they use your program. Error handling improves your script functionality by managing errors as they happen and creating processes of recovering. Most modern languages use the try-except construct to catch errors, Python included. When using this method, you can create simple user-friendly error messages should an error occur. Another advantage is the ability to apply customized error handling to grouped statements.

Exception Class

In this assignment we used the try-except construct to handle any errors that may arise. The exception class holds information about the error that just occurred while the code was running. In Python, the Exception class will automatically create an Exception object when the error occurs. The Exception object will provide information about the error that causes the exception. This is useful because we can provide both the simple and complex error messages for the user as shown below.



```
if menu_choice == "1":  # This will not work if it is an integer!

    try:
        student_first_name = input("Enter the student's first name: ")
        # ErrorHandling
        if not student_first_name.isalpha():
            raise ValueError("ERROR: The student's first name must contain alphabetical characters.")
        student_last_name = input("Enter the student's last name: ")
        # ErrorHandling
        if not student_last_name.isalpha():
            raise ValueError("ERROR: The student's last name must contain alphabetical characters.")
        course_name = input("Please enter the name of the course: ")
        student_data = {"FirstName": student_first_name,
                        "LastName": student_last_name,
                        "CourseName": course_name}
        students.append(student_data)
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as e:
        print(e) #prints custom message
        print("-- Technical Error Message --")
        print(e.__doc__)
        print(e.__str__())
```

```
----------------------------------------

What would you like to do: 1
Enter the student's first name: fe3wf435
ERROR: The student's first name must contain alphabetical characters.
-- Technical Error Message --
Inappropriate argument value (of correct type).
```

Figure 6: Generic and Technical Error Message

Summary

In summary, I found this assignment to be the most difficult thus far. I learned about the differences between JSON and CSV files, using dictionaries and lists. The concept and convenience of try-except error handling but it seems like a significant amount of additional code. I think it would be beneficial to have a syntax cheat sheet in cases like this.