Mia Bolding

07/27/2024

Introduction to Programing with Python

Assignment 05 – Programming Basics

Python Dictionary JSON and Exception

Introduction

This week, I learned about Python Dictionaries, JSON, and Exception Handling. A dictionary in Python stores data as key-value pairs, making data easy to access and modify. JSON is a format for structuring data, often used to exchange information between web applications. Python's json module helps convert data between dictionaries and JSON. Exception handling in Python uses try, except, else, and finally blocks to manage errors gracefully, ensuring the program continues running smoothly even when something goes wrong.

Dictionary and JSON

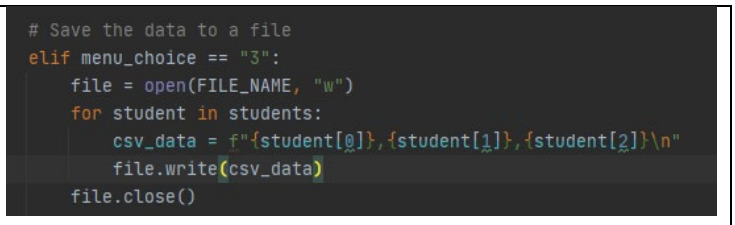
In these sections, In Assignment 05, I use dictionaries to structure and manage the student data, storing each student's first name, last name, and course in a key-value format. The list of dictionaries (students) allows for easy manipulation and storage of multiple student records.

Compared to the JSON and list formats, JSON format is much easier to read and quicker to write. See the example below for JSON format for students' records from Assignment 05.

[{"first_name": "Jana", "last_name": "Demas", "course": "Python 110"}, {"first_name": "Evan", "last_name": "Inkster", "course": "Python 110"}, {"first_name": "Hanh", "last_name": "Bui", "course": "Python 210"}]

Lesson Learn During Assignment

When I switched from using a list of lists to a list of dictionaries, I encountered a KeyError: 0 when trying to write data to the CSV file. This error occurred because I used numeric indices to access the dictionary values instead of using the appropriate dictionary keys. After identifying the issue, I corrected the code by using the dictionary keys to access the values and format the csv_data string properly for writing to the CSV file. Now my Enrollments.csv saves the data input as 'Jana Demas is enrolled in Python 110'. See **figure 1 and 2.**

Figure 1. Codes cause KeyErrorError message	
Figure 2. Access dictionary values using keys	

```
# Save the data to a file
elif menu_choice == "3":
    file = open(FILE_NAME, "w")
    for student in students:
        csv_data = f"{student[0]},{student[1]},{student[2]}\n"
        file.write(csv_data)
    file.close()
```

```
# Save the data to a file
elif menu_choice == "3":
    file = open(FILE_NAME, "w")
    for student in students:
        csv_data = f"{student['first_name']} {student['last_name']} is enrolled in {student['course_name']}\n"
        file.write(csv_data)
    file.close()
```

I find using exceptions to manage high-risk code lines or potential errors quite challenging. The challenging part for me is handling indentations. Many times, I find myself misplacing the indentations, which causes syntax errors or results in the program executing without a few lines of code. However, having exceptions in place is essential for handling errors during program execution. They help avoid unpleasant redline warnings and prevent the program from stopping abruptly.

To handle a FileNotFoundError in Assignment 05, I first place the code that might raise the error inside a try block. If a FileNotFoundError occurs, I use the except block to catch it and print a message like "File not found" to inform the user. Finally, I use the finally block to ensure the file is closed if it was opened, regardless of whether an error occurred. **Figure 3.**

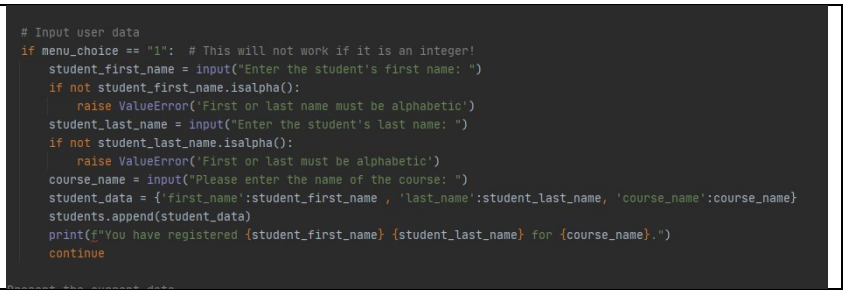
Figure 3. Handle Exception in File	

```
try:
    file= open(FILE_NAME, "r")
    for row in file.readlines():
            # Transform the data from the file
            parts = row.strip().split(',')
            student_first_name = parts[0]
            student_last_name = parts[1]
            course_name = parts[2]
            # Setting up keys
            student_data = {'first_name': student_first_name, 'last_name': student_last_name, 'course_name': course_name}
            # Load it into our collection (list of dictionaries)
            students.append(student_data)
except FileNotFoundError:
    print('File not found. Creating..')
    open(FILE_NAME, 'w')
    file.close()
except Exception as e:
    print('Unknown exception', type(e), e, sep='\n')
finally:
    if not file.closed:
        file.close
```

How to Fix Errors with User Input

Similar to handling file exceptions, I anticipate that users might accidentally enter non-alphabetic characters when typing their names. To address this, I use if not student_last_name.isalpha() to check if the input contains non-alphabetic characters. If this condition is met, I raise a ValueError with the message "First or last name must be alphabetic" to provide a gentle reminder. **Figure 4**

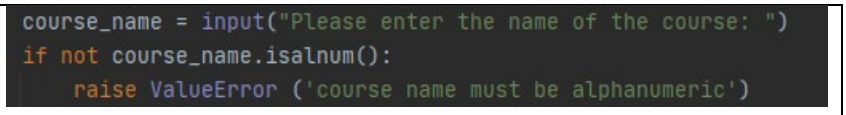
Figure 4. Handle Exception in User Input – alphabetical value



```
# Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    student_first_name = input("Enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError('First or last name must be alphabetic')
    student_last_name = input("Enter the student's last name: ")
    if not student_last_name.isalpha():
        raise ValueError('First or last must be alphabetic')
    course_name = input("Please enter the name of the course: ")
    student_data = {'first_name':student_first_name , 'last_name':student_last_name, 'course_name':course_name}
    students.append(student_data)
    print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    continue
```

As shown above **Figure 4**, I do not set up any exception for the course name since course_name is an alphanumeric value. Instead, I use the exception rule if not course_name.isalnum(): to reflect the nature of this value.

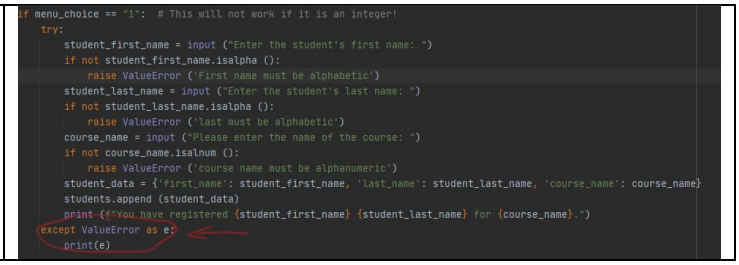
Figure 5. Handle Exception in User Input - alphanumeric value



```
course_name = input("Please enter the name of the course: ")
if not course_name.isalnum():
    raise ValueError ('course name must be alphanumeric')
```

Finally, if the exceptions I set up catch any issues—such as a first or last name not being alphabetic or a course name not being alphanumeric—I display the message to the user by naming these exceptions as e and printing e to show the error. One important thing I've learned is to ensure proper indentation for the try and except blocks so that Python can correctly interpret my code. See Figure 6 for reference. **Figure 6.**

Figure 6: Catch Exception and Display Message



```
if menu_choice == "1":  # This will not work if it is an integer!
    try:
        student_first_name = input ("Enter the student's first name: ")
        if not student_first_name.isalpha ():
            raise ValueError ('First name must be alphabetic')
        student_last_name = input ("Enter the student's last name: ")
        if not student_last_name.isalpha ():
            raise ValueError ('last must be alphabetic')
        course_name = input ("Please enter the name of the course: ")
        if not course_name.isalnum ():
            raise ValueError ('course name must be alphanumeric')
        student_data = {'first_name': student_first_name, 'last_name': student_last_name, 'course_name': 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)
```

Summary

I use Python dictionaries to store data in key-value pairs, which allows me to efficiently access and update information. JSON helps me convert these dictionaries to and from strings, making it easier to share data across different systems. Exception handling lets me manage errors gracefully with try and except blocks, ensuring my program runs smoothly and handles unexpected issues effectively.