Katherine Sather

11/10/2024

Foundations of Programming: Python

Assignment No. 5

Assignment No. 5

# Introduction

In module five, I expanded my programming skills by diving into key concepts like GitHub, JSON, and dictionaries. I also explored structured error handling, particularly using the “try-except” method. This module's assignment challenged me to integrate these elements, resulting in a more sophisticated and robust version of my code.

# Writing the program

For assignment five, I built a course registration system that demonstrates using constants, variables, dictionaries, and structured error handling. It allows users to register students for a course, display current registrations, and save the data to a JSON file.

Since we’re working with JSON, I added this code to the top of my file: import json. This line imports Python's json module, which provides functionality to work with JSON (JavaScript Object Notation) data. JSON is commonly used for storing and exchanging data between systems. For what it’s worth, I was curious when to use “json” vs the uppercase version, and this is what I learned:

1. **When referring to the data format itself (as a standard)**, you typically capitalize it: **"JSON"** (e.g., "JSON is commonly used for data exchange").
   * This is because "JSON" is an acronym for **JavaScript Object Notation**, so it’s treated like other acronyms.
2. **When referring to code-related items** such as module names, functions, or methods, you use lowercase: **json**.
   * For example, Python’s json module and functions like json.load() or json.dump() are all lowercase.

Next I began defining the data. Here are the items of particular interest: I defined student\_data as a dictionary, with curly brackets. It will serve to store a single student's information (first name, last name, and course name). I defined **students** as a list, with square brackets. Each student’s data is expected to be a dictionary added to this list, creating a collection of student records (See Fig 1).

A screenshot of a computer code

Description automatically generated

***Figure 1: Importing json and defining data.***

## Read the current program content

In the next section of my code, you’ll see instructions for starting the program by reading the contents of "Enrollments.json" (see Fig. 2). Although this code appears after the data definitions, I added it later, so the program’s structure doesn’t necessarily reflect the exact order I followed while working on the assignment. This code includes the “try-except” method, and so begins with “try:”. Next I tell the program to open “Enrollments.json” for reading, then load the JSON file. Using a “for” loop, I instruct the program to display content for each student in the list of students, asking it to print the dictionary key-value pairs established in another step.

Next I add this code: *except Exception as e*. This line is an example of “structured data handling.” It catches any exception that occurs in the preceding try block and assigns it to the variable e. This allows the program to handle all types of exceptions in a general way. I customized the error message to inform users of possible issues, such as the file not existing or containing invalid data (I couldn’t think of any other type of error scenario). So my error message reads: “Something went wrong. Make sure the file exists and has valid json content and try again.” (see Figure 2).

A screenshot of a computer code

Description automatically generated

***Figure 2: Instructing the program to start by displaying file content and adding instructions for handling errors.***

## Input and output

Next in the code, I instruct the program on how to proceed if a user selects options 1 or 2 from the menu. Option 1 lets users register students for a course, entering a first name, last name and course name. This information is ultimately stored as a dictionary (student\_data) and appended to a list (students). Another “try-except” pattern is included here, to handle any input issues.

Selecting option 2 allows users to view current data. To accomplish this, I used the print function and a “for” loop that tells the program to reference key-value pairs for each student. In this code, I used an f-string to tell the program that for printed content outside of curly braces, use a string literal as the format (See Fig 3).

A close-up of a computer screen

Description automatically generated

***Figure 3: Program instructions for menu option 2.***

# Saving the data

Selecting option 3 from the menu allows the user to save student data to the JSON file. This section again starts a try block to handle any exceptions that might occur. We tell the program to open “Enrollments” in write mode, overwriting any existing content in the file. The code “json.dump” writes contents “students” to the file in JSON format. I close the file to ensure everything saved and then reopen it to read and confirm content. (see Fig 4).

A screenshot of a computer program

Description automatically generated

***Figure 4: Saving the new student data in json format and then printing it.***

# Summary

In Assignment 5, I created a course registration system that deepened my understanding of GitHub, JSON, dictionaries, and error handling with try-except. The program allows users to register students, display registrations, and save data to a JSON file.

I imported the json module to manage JSON data, used a dictionary (student\_data) to hold individual student details, and stored multiple records in a list (students). The program begins by reading from "Enrollments.json" with error handling to catch missing or invalid files. Menu options let users add or view registrations, and a final option saves data to the JSON file. This assignment taught me to integrate data processing and error handling for a more robust program.