Hyrum Magnusson

2025FEB26

IT FDN 110 A Wi 25: Foundations of Programming: Python

Assignment 05

https://github.com/HMag-PFun/IntroToProg-Python

Advanced Collections and Error Handling

Introduction

This weeks module explores advanced data collection methods and structured error handling techniques in Python. Specifically, we delve into dictionary collections, working with JSON files, and implementing structured error handling to ensure data integrity and smooth program execution.

Dictionary Collections

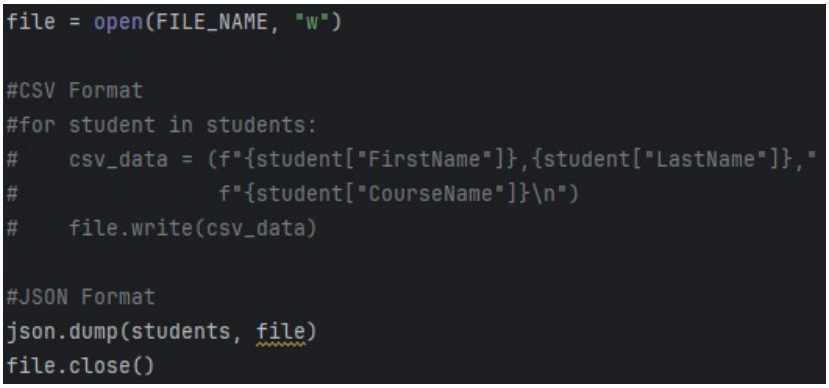
Dictionaries are a fundamental data structure in Python, consisting of key-value pairs where each value is associated with a unique key. Primarily used for mapping and retrieving data efficiently. Like sets, dictionaries are designated by curly braces {} separating elements with commas. Though each element is a key-value pair separated by a colon : to differentiate it as a dictionary in Python.

Dictionaries offer great flexibility and mutability, allowing key-value pairs to be dynamically added, modified, or removed. This versatility makes dictionaries able to grow and evolve as the database expands. The efficient lookup of values via unique keys allows for quick retrieval, modification, and deletion of specific data entries. This adaptability makes dictionaries a powerful tool in data analysis, database management, and application development.

Working with JSON files

JSON (JavaScript Object Notation) is a lightweight, text-based file format commonly used for data exchange between application. The minimal formatting and human readability of this cross language file structure makes it effective at organizing, storing, and transmitting data between different applications Figure 1 shows how code is simplified when using JSON file.

Figure 1: *Example of simplified code when using JSON over CSV*



```
file = open(FILE_NAME, "w")

#CSV Format
#for student in students:
#    csv_data = (f"{student["FirstName"]},{student["LastName"]},"
#                f"{student["CourseName"]}\n")
#    file.write(csv_data)

#JSON Format
json.dump(students, file)
file.close()
```

Advantages of JSON:

- Cross-language compatibility: JSON is widely supported across different programming languages, making it an ideal choice for data interchange.
- Human-readable format: The structured yet simple syntax allows for easy readability and debugging.
- Hierarchical data representation: JSON supports nested structures, making it suitable for complex data storage needs.

While JSON has many advantages over CSV files in handling structured and hierarchical data, CSV files remain preferable for large, tabular datasets due to their simplicity and performance benefits. However, CSV files are more prone to data integrity issues due to the lack of strict formatting.

Structured Error Handling

Error handling is a process for which the programmer can anticipate and direct how the program responds to unexpected issues. This is importing for keeping the program running smoothly without crashing giving ways to detect and resolve issues. This is especially useful for data integrity when asking for user input, error handling allows to restrict certain responses from the user such as a name containing a number or special characters or receiving a letter when requesting an integer value. In Python this is accomplished with a try-except construct, which allows for custom error messages to direct the end user on how to resolve the issue

Figure 2: *Example a custom error handler.*

```
try:
    student_first_name = input("Enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError("The first name should not contain numbers")
    student_last_name = input("Enter the student's last name: ")
    if not student_last_name.isalpha():
        raise ValueError("The last name should not contain numbers")
    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} "
          f"{student_last_name} for {course_name}.")
    continue
except ValueError as e:
    print(e)  # Prints the custom message
    print('Built-In Python error info: ')
    print(e, e.__doc__, type(e), sep='\n')
```

Summary

In this module, we explored advance data collections and structured error handling techniques in Python. Dictionaries provide a flexible and efficient way to store and retrieve data, while JSON files facilitate cross-platform data exchange. Additionally, structured error handling using try-except constructs ensures robust program execution by managing errors effectively. Mastering these concepts is essential for developing reliable and scalable software applications.