Mariam Assaad
Wednesday, November 13, 2024
IT FDN 100 A - Foundations of Programming: Python
Assignment 05

# Data Management and Error Handling in Python Programming: Working with Dictionaries, JSON, and GitHub

## Introduction

In this module, I expanded my Python programming skills by exploring advanced data management techniques, structured error handling, and essential tools for collaborative coding. The module covers the practical application of dictionaries for structured file data storage, working with JSON files for handling complex data formats, implementing try/except blocks to manage errors, and using GitHub for code versioning and sharing. Each section of this document provides insights into these concepts, demonstrating how they enhance code efficiency, reliability, and collaboration. By mastering these tools, I am equipped to handle data-driven programming challenges with a professional approach.

## 1. Using Dictionaries with Files

In this section, I learned how to organize and store data using Python dictionaries and write it to a file. Dictionaries allow data to be stored as key-value pairs, making it easy to retrieve specific information. I explored creating a data table using a list of dictionaries, adding user input as new entries, and saving this structured data to a text file. This approach enables efficient data storage and retrieval in Python, especially when handling organized data sets.

The code below demonstrates the process of creating a list of dictionaries, each representing a department with specific attributes (major name, department, and location):

```
# Initializing the departments with existing major details
department_row1:dict[str,str] = {'major_name':'Mechanical',
'dept_name':'Engineering', 'location':'Seattle'}
department_row2:dict[str,str] = {'major_name':'English',
'dept_name':'Literature', 'location':'Seattle'}
department_row3:dict[str,str] = {'major_name':'Business',
'dept_name':'Technology', 'location':'Seattle'}

# Creating a list to store all department dictionaries
departments:list[dict[str,str]] = [department_row1, department_row2,
department_row3]

# Loop through departments and print each major, department, and location
for row in departments:
    print(f'{row["major_name"]} {row["dept_name"]} major is available in
{row["location"]}')

# Adding a new major based on user input
```

```python
# Prompt the user for major name, department name, and location
university_major_name:str=input('Please enter the name of the major: ')
major_department_name:str=input('Please enter the name of the department: ')
university_location:str=input('Please enter the name of the location: ')

# Creating a dictionary for the new major using user input
department_row4 = {'major_name': university_major_name, 'dept_name':
major_department_name, 'location': university_location}

# Adding the new major to the departments list
departments.append(department_row4)

# Loop through updated departments list and print details of each major,
including the newly added one
for row in departments:
    print(f'{row["major_name"]} {row["dept_name"]} major is available in
{row["location"]}')
```
**Figure 1: Adding and Displaying University Majors Using Dictionaries in Python**

The code in Figure 1 demonstrates how to manage university majors using dictionaries in Python. It begins by creating dictionaries to represent each major, department, and location. User inputs are then collected to dynamically add a new major to the list, illustrating how Python's dictionary structure supports the addition of key-value pairs. The code also shows how to append each new entry to a list of dictionaries, and how to display each entry with formatted output, making it clear and organized.

## 2. Using JSON Files

This section introduced JSON (JavaScript Object Notation) files, a widely-used format for data interchange. JSON structures data in a readable format similar to Python dictionaries, making it ideal for data exchange and storage in web applications, configuration files, and more. JSON's flexibility supports various data types, such as strings, numbers, arrays, and nested objects, allowing complex data structures to be represented hierarchically. Below is an example of a JSON file, which contains data about university majors. Each major is represented as an object within an array, with attributes for the major name, department, location availability, and required entry GPA.

```json
[
  {
    "MajorName": "Mechanical",
    "DepartmentName": "Engineering",
    "InSeattle": true,
    "EntryGPARequired": 3.7
  },
  {
    "MajorName": "English",
    "DepartmentName": "Literature",
    "InSeattle": true,
    "EntryGPARequired": 3.0
  }
]
```
**Figure 2: Example JSON Structure Representing University Majors**

Content Breakdown:

- The file contains an array of objects (enclosed in square brackets []).
- Each object (enclosed in curly braces {}) represents a department major and includes the following fields:
    - "MajorName": The name of the major (e.g., "Mechanical" or "English").
    - "DepartmentName": The department associated with the major (e.g., "Engineering" or "Literature").
    - "InSeattle": A boolean value (true or false) indicating if the department is available in Seattle.
    - "EntryGPARequired": A numeric value representing the GPA required for entry.

This format is particularly useful when programs need to load, process, and display structured data, such as academic records. With Python's json module, each entry can be read and manipulated individually, making JSON a practical choice for data handling in educational and web-based contexts.

## 3. Try/Except

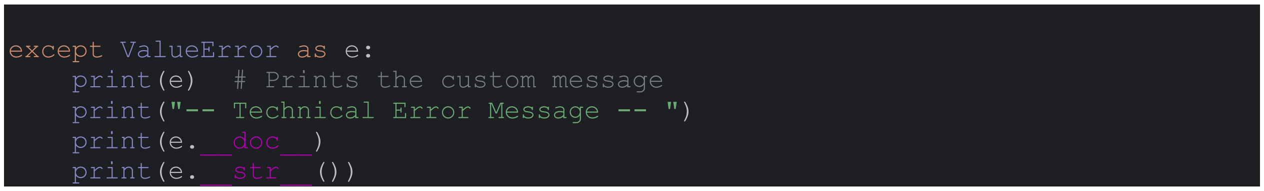
In this section, I explored error handling in Python using the try/except construct. This approach helps catch and manage errors in code execution, improving user experience by providing custom error messages instead of unexpected program crashes. The following code snapshot demonstrates how input is collected for a new department, with specific checks in place for each input type. If a validation error occurs, a custom error message is displayed, guiding the user to correct their input. This code showcases robust data handling practices that enhance reliability by catching and addressing common input errors.

```
try:
    # Check that the input does not include numbers
    major_name = input("Enter the name of the major: ")
    if not major_name.isalpha():
        raise ValueError("The major name should not contain numbers.")

    department_name = input("Enter the name of the department: ")
    if not department_name.isalpha():
        raise ValueError("The department name should not contain numbers.")

    in_seattle = input("Is this major available in Seattle? (true/false):
").lower() == 'true'
    entry_gpa = input("Enter the entry GPA required: ")
    if not entry_gpa.replace('.', '', 1).isdigit():
        raise ValueError("The GPA should be a numeric value.")

    department_table.append({
        "MajorName": major_name,
        "DepartmentName": department_name,
        "InSeattle": in_seattle,
        "EntryGPARequired": float(entry_gpa)
    })
```



```
except ValueError as e:
    print(e)  # Prints the custom message
    print("-- Technical Error Message -- ")
    print(e.__doc__)
    print(e.__str__())
```

**Figure 3: Validating and Handling Errors in User Input for Department Data**

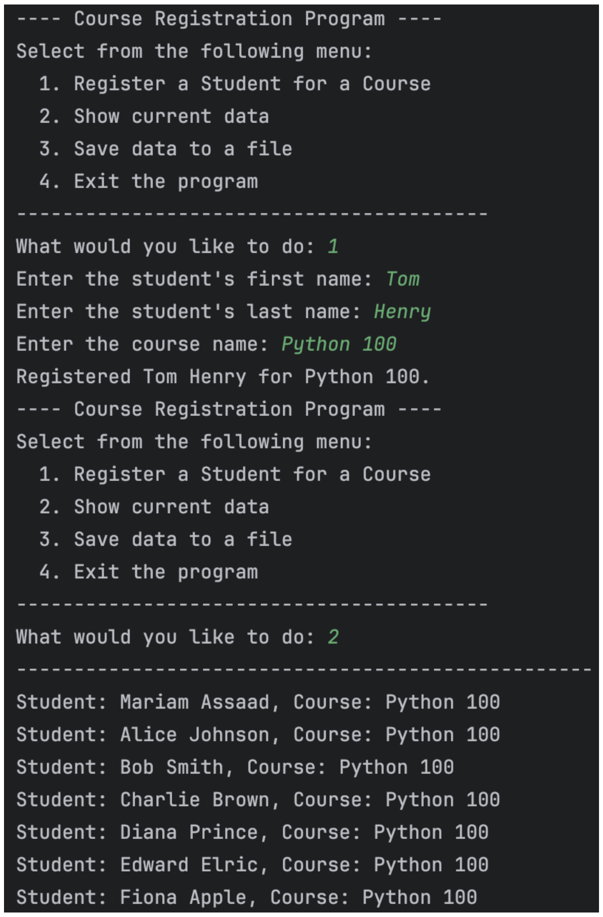
In Figure 3, we see how the program ensures that each field is properly validated before appending the new department entry to the department_table. Each input prompt is followed by validation logic to check for alphabet-only entries in names and numeric values for GPA, which are critical for maintaining data integrity. The ValueError exception is raised when invalid input is detected, prompting the user with a custom error message and technical details. This setup enhances user experience by providing clear feedback while protecting the program from incorrect data entries.

## 4. Using the GitHub Website

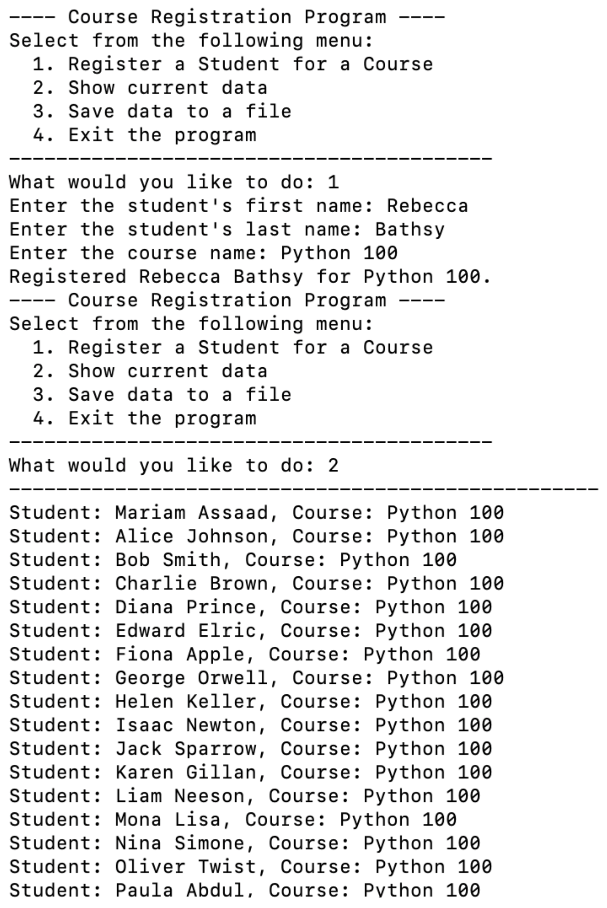
This section covered GitHub, a platform for version control and collaboration on code projects. I learned to create a GitHub account, set up a repository, and understand the basics of managing project files on the platform. GitHub repositories provide an organized way to store, track, and share code with others, making it easier to collaborate on projects. This experience with GitHub prepares me to work in team environments and utilize industry-standard version control practices. I created a repository for Assignment 05, where I uploaded all the code and files from this module. You can view the repository and project files here:

https://github.com/mariam-assaad/IntroToProg-Python-Mod05

## Testing and Running Assignment 5



```
---- 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
-----------------------------------------
What would you like to do: 1
Enter the student's first name: Tom
Enter the student's last name: Henry
Enter the course name: Python 100
Registered Tom Henry for Python 100.
---- 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
-----------------------------------------
What would you like to do: 2
--------------------------------------------------
Student: Mariam Assaad, Course: Python 100
Student: Alice Johnson, Course: Python 100
Student: Bob Smith, Course: Python 100
Student: Charlie Brown, Course: Python 100
Student: Diana Prince, Course: Python 100
Student: Edward Elric, Course: Python 100
Student: Fiona Apple, Course: Python 100
```



```
---- 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
-----------------------------------------
What would you like to do: 1
Enter the student's first name: Rebecca
Enter the student's last name: Bathsy
Enter the course name: Python 100
Registered Rebecca Bathsy for Python 100.
---- 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
-----------------------------------------
What would you like to do: 2
--------------------------------------------------
Student: Mariam Assaad, Course: Python 100
Student: Alice Johnson, Course: Python 100
Student: Bob Smith, Course: Python 100
Student: Charlie Brown, Course: Python 100
Student: Diana Prince, Course: Python 100
Student: Edward Elric, Course: Python 100
Student: Fiona Apple, Course: Python 100
Student: George Orwell, Course: Python 100
Student: Helen Keller, Course: Python 100
Student: Isaac Newton, Course: Python 100
Student: Jack Sparrow, Course: Python 100
Student: Karen Gillan, Course: Python 100
Student: Liam Neeson, Course: Python 100
Student: Mona Lisa, Course: Python 100
Student: Nina Simone, Course: Python 100
Student: Oliver Twist, Course: Python 100
Student: Paula Abdul, Course: Python 100
```

**Figure 5: The program runs correctly in both PyCharm and from the console or terminal.**

## Summary

This module introduced essential Python programming techniques for handling data, managing errors, and using collaborative tools. By exploring dictionaries, JSON files, and file manipulation, I gained practical skills for organizing and storing structured data. The section on try/except blocks provided valuable insight into error management, allowing for smoother and more user-friendly code execution. Additionally, I learned to utilize GitHub's platform for version control, facilitating efficient project collaboration. Together, these concepts lay a strong foundation for managing data, handling errors, and enhancing collaboration in real-world programming environments.