Marina Osiechko

February 23, 2026

Foundations Of Python Programming

Assignment 05

Python Programming Tools and Techniques

Introduction

This paper describes the steps I followed to complete assignment five in the ***Introduction to Programming with Python*** course. This module focused on fundamental programming concepts, including data structures, file handling, and structured error management. I learned how to write data from dictionaries into a file and how to reverse the process, by loading data back into a file. I was also introduced to JavaScript Object Notation (JSON), version control practices, and the importance of GitHub in modern software development. Overall, this assignment demonstrated how to use dictionaries, files, and exception handling to build a functional course-registration program.

Task 1: Defining Data Constants and Variables

The assignment began with importing the ***json module*** and defining the program's **constants and variables.** The constant **MENU** displayed the list of user choices using a multi-line string create with triple quotes. As required, constants were written in uppercase and remained unchanged throughout the program.

Variables were written in lowercase and were designed to store user input or program data. The variables ***student_first_name***, ***student_last_name***, ***course_name,*** and ***menu_choice*** were initialized as empty strings. The **students** variable was defined as data type list, and ***student_data*** was defined as a dictionary.

In this module, I learned how to work with dictionaries to store structed data. The ***student_data*** dictionary formatted each registration record, and each dictionary was appended to the ***students*** list, which served as a two-dimensional list of dictionary rows. Figure 1 shows the code used to define constants and variables:

```
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

  ——————————————————-''')
```

```
FILE_NAME: str = "Enrollments.json"  # Caps signify constants

# Define the Data Variables

student_first_name: str = ""  # str = "" stands for empty string

student_last_name: str = ""

course_name: str = ""

file = None

menu_choice: str = ""

student_data: dict = {}  # one row of student data dictionary (which is one row)

students: list = [ ] # two-dimensional list of dictionary rows, a table of students data
```

Figure 1: Defining the Data Constants and Variables

Task 2: Formatting and Creating the JSON File

To prevent errors when the program first loads, the assignment required an existing JSON file containing student information. I created an initial ***Enrollments.json*** file by defining two sample student dictionaries and writing them to the file using ***json.dump().*** Once the file was created, the code was commented out to avoid overwriting future registrations. Figure 2 demonstrates the code I used to create a new JSON file:

```
# Formatting names to add to the newly created dictionary
student_row1: dict[str, str] = {"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}
student_row2: dict[str, str] = {"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}
students: list[dict[str, str]] = [student_row1, student_row2]

# Create the file (or overwrite if exists)
file = open('Enrollments.json', 'w')
json.dump(students, file, indent=2)  # Write the Python data into the file as JSON
file.close()
print('Data Saved!')
```

Figure 2: Creating JSON file

Task 3: Loading Existing Data into the Program (Dictionary)

When the program starts, it automatically reads the contents of ***Enrollments.json*** into the ***students*** two-dimensional list of dictionary rows. This ensures that previously saved registrations appear in the program before any new data is added. The file-loading process includes structured error handling to manage missing files, invalid formats, or unexpected exceptions. A ***finally*** block ensures that the file is properly closed. Figure 3 shows the code for loading existing data from the ***Enrollemnts.json*** into ***students*** list of dictionary rows and the structured error handling:

```
# When the program starts, read the file data into a list of dictionary rows

try:

  # Open JSON file, dump data into students table and closes (load converts JSON into Python)

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

```
students = json.load(file)
except FileNotFoundError as e:
  print('Test file must exist before running this script!\n')
  print('-- Technical Error Message --')
  print(e, e.__doc__, type(e), sep='\n')
except Exception as e:  # this is general Catch All error exception
  print('There was a non-specific error!\n')
  print('-- Technical Error Message --')
  print(e, e.__doc__, type(e), sep='\n')
finally:
  # Check if a file object exists and is still open
  if file is not None and file.closed == False:
    file.close()
```

Figure 3: Loading Existing Data to a List of Dictionary Rows

Task 3: Presenting and Processing User Input
The main section of the program uses a ***while True*** loop to repeatedly display the menu and process the user's selection. The program uses ***if/elif*** statements to handle the four menu choices corresponded to the options shown in Figure 1.

Menu choice one prompts the user to enter a first name, last name, and course name. Input validation ensures that names contain only alphabetic characters. Valid data is stored in a dictionary and appended to the ***students*** list of dictionary rows by the ***append()*** function. Structured error handling manages invalid input and unexpected exceptions. Figure 4 shows the code for menu choice one:

```
# Present and process the data
while (True):


  # Present the menu of choices
  print(MENU)
  menu_choice = input('What would you like to do: ')


  # Input user data
  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('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: ')

        # Collected data is added to a dictionary named student_data
        student_data = {
            "FirstName": student_first_name,
            "LastName": student_last_name,
            "CourseName": course_name
        }
        # Student_data is added to the students 2D list of dictionary rows
        students.append(student_data)
        print(f'You have registered {student_first_name} {student_last_name} for {course_name}.')
    except ValueError as e:
        print(e)  # Print the custom message
        print('— Technical Error Message —')
        print(e.__doc__)
        print(e.__str__())
    except Exception as e:
        print('There was a non-specific error!\n')
        print('— Technical Error Message —')
        print(e, e.__doc__, type(e), sep='\n')
    continue
```

Figure 4: Input User Data

Menu choice two prints all stored registration records as coma-separated values. Because the program stores multiple dictionaries in the students list, it can display multiple registrations at once. Figure 5 shows the code used for choice two:

```
  # Present the current data
  elif menu_choice == '2':
    # Process the data to create and display a custom message
    print("-" * 50)
    for student_data in students:

print(f'{student_data["FirstName"]},{student_data["LastName"]},{student_data["CourseName"]}')
    print("-" * 50)
    continue
```

Figure 5: Presenting the Data

When the user selects choice three, the program opens ***Enrollments.json*** in write mode and uses ***json.dump()*** function to save the entire ***students*** list. The file was closed by using the ***close()*** method and displayed the saved data. Error handling ensures that invalid data or write failures are properly reported. After saving, the program prints the data that was written to the file. Figure 6 shows the file-processing code for menu choice three:

```
  # Save the data to a file
  elif menu_choice == '3':
    # Load student data into JSON (list of dictionaries) table (dump changes Python to JSON)
    try:
      file = open(FILE_NAME, 'w')
      json.dump(students, file, indent=2)
    except TypeError as e:
      print('Please check that the data is a valid JSON format\n')
      print('– Technical Error Message –')
      print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
      print('– Technical Error Message –')
      print('Built-In Python error info: ')
      print(e, e.__doc__, type(e), sep='\n')
    finally:
      if file is not None and file.closed == False:
        file.close()
    print('The following data was saved to file!')
    for student in students:
      print(
        f'Student {student["FirstName"]} {student["LastName"]} is enrolled in '
        f'{student["CourseName"]}'
      )
    continue  # the loop
```

Figure 6: Saving the Data

The final option breaks the loop and ends the program. After the loop exits, a final message confirms that the program has completed. Figure 7 shows the code used for menu choice four:

```
    # Stop the loop
    elif menu_choice == "4":
      break        # This exits the program

    else:

          print("Please only choose options 1, 2, 3, or 4!")

print("Program Ended")     # This line runs after the loop finishes
```

Figure 7: Exiting the Program

Task 4: Code Testing
I tested the program by running it and confirming that it accepted user input, displayed existing file contents, formatted data into comma-separated strings, and saved output to a JSON file. The program successfully handled multiple registrations and ran correctly in both PyCharm and the terminal.

Task 5: Posting to GitHub
As part of this assignment, I uploaded my homework files to a public GitHub repository so they could be reviewed by others. I created a repository titled ***IntroToProg-Python-Mod05*** and posted all required files there. The link to my repository is included for reference: MarinaOsiechko/IntroToProg-Python-Mod05.

Summary

In this module, I strengthened my understanding of Python data structures, especially dictionaries and lists, and learned how to store and retrieve data using JSON files. I practiced structured error handling to make the program more reliable and user-friendly. I also gained experience with file operations, menu-driven program design, and validating user input. Completing this assignment helped me build a functional course-registration system and deepened my confidence in writing Python programs that manage real-world data.