Shawn Kennedy
02/21/2026
Foundations of Programming Python
Assignment 05
GitHub URL: https://github.com/TheeSPK7/IntroToProg-Python-Mod05.git

Introduction

Week 5 introduced several foundational concepts that distinguish functional code from production-ready applications: dictionaries for structured data storage, JSON for standardized file formatting, try/except blocks for structured error handling, and GitHub for version control and collaboration. These tools collectively enable developers to build programs that perform reliably even when faced with corrupted files, invalid user input, or unexpected system states. As someone accustomed to the precision required in federal compliance reporting and database management, I appreciated how these Python techniques mirror the robust error handling and data validation strategies I use daily. This week challenged me to think beyond "does it work?" and toward "what happens when something goes wrong?"

Working with Dictionaries and Files (Lab 1)

A dictionary in Python is a collection of key-value pairs. Each key is connected to a value, and you can use a key to access the value associated with that key. A key's value can be a string, a number, a list, or even another dictionary. We use {} braces to identify the dictionary with a series of key-value pairs inside.
In the first lab this week, we transformed data from a list format into a dictionary. After reading and splitting a CSV row into a list, we reassigned student_data as a dictionary:

```
student_data = {"FirstName": student_data[0],
                "LastName": student_data[1],
                "GPA": float(student_data[2].strip())}
```

This creates a mapping between descriptive keys ("FirstName", "LastName", "GPA") and their corresponding values from the file. This dictionary structure makes the code more readable, instead of remembering that index [0] is the first name, we can simply use student_data["FirstName"], which is much more explicit.

Working with JSON files (Lab 2)

JavaScript Object Notation (JSON) was introduced this week as a data storage option. Since JSON is not *native* to Python, it must be imported: import json
Key components for file-based JSON operations are json.dump() and json.load(). The json.dump() function writes Python data structures (like lists and dictionaries) directly to a file in JSON format. The json.load() function reads JSON data from a file back into Python data structures.
In the second lab, we replaced the CSV file reading approach with JSON. To read the stored student data:

```
file = open(FILE_NAME, 'r')
students = json.load(file)
file.close()
```

This single json.load() call replaces the previous approach of reading line-by-line, splitting on commas, and manually converting data types. JSON automatically preserves data types (strings, numbers, lists, dictionaries) and structure.
When displaying student data, we access dictionary values using keys:

```
print(message.format(student["FirstName"], student["LastName"], student["GPA"]))
```

To save the updated student data back to the file:

```
file = open(FILE_NAME, 'w')
json.dump(students, file)
file.close()
```

The json.dump() function automatically converts our Python list of dictionaries into properly formatted JSON text and writes it to the file.

Python also provides json.dumps() and json.loads() for working with JSON as strings in memory rather than files, though we don't use those in this lab.

Working with Exceptions (Lab 3)

One of the most valuable concepts we learned this week was exception handling. Learning how to handle errors so that the program does not crash during unexpected situations is pivotal for any Python developer. The challenging part for me was mastering the formatting and indentation, which is critical for exception blocks to work correctly. In this lab, I encountered indent error after indent error and redid the lab multiple times just to develop a better grasp. This repetition proved worthwhile when working on the assignment, where I felt much more comfortable.
After defining the constants and variables, the first exception handler was written:

```
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
except FileNotFoundError as e:
    print("Text file must exist before running this script!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if file is not None and file.closed == False:
        file.close()
```

Initially, I misunderstood the structure and thought each exception type required its own separate try/except block. After rewatching the videos and taking better notes, I learned that a single try block can have multiple except clauses to handle different error types sequentially. The order matters - Python checks each except clause from top to bottom and executes the first one that matches.
Two important exception types we used were TypeError and ValueError. A TypeError is raised when an operation is applied to an object of an inappropriate or incompatible

data type, while a ValueError occurs when a function receives an argument of the correct type but an inappropriate value. For example:

```
except ValueError:
        print("GPA must be a numeric value.")
except TypeError as e:
        print("Please check that the data is in valid JSON format\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
```

After implementing exception handling throughout the script and testing various error scenarios, I achieved the desired results and gained a solid understanding of how exception handling prevents program crashes and provides meaningful error messages to users.

Saving files to GitHub (Lab 04)

I had previously created a personal GitHub account in a prior course with Mr. Root, though it has remained unused since then. I primarily use my work GitHub account, which is used almost daily to share information between teams located around the state. For this lab, I uploaded my Mod05-Lab03 Python script to my personal GitHub repository at https://github.com/TheeSPK7/Python110-Winter2026.

Maintaining a GitHub repository for coursework will be valuable as we progress through the course. It provides version control to track changes in our code, creates a portfolio of development work, and facilitates collaboration and code sharing with classmates and instructors.

Building Assignment 5

Assignment 5 required creating a Course Registration Program that incorporated all the elements from the week's labs - dictionaries, JSON file handling, and exception handling - culminating with uploading the final product and this write-up to GitHub.
The most challenging part of this assignment was the "Input User Data" section (menu choice 1). This portion of the code integrated every element from Labs 1-3: dictionary creation, input validation, and exception handling. My biggest struggle remained proper indentation and the correct structure of nested try/except blocks. However, the time I spent working through Lab 3 multiple times paid off, and I was able to implement the code correctly:

```
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: ")
        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} {student_last_name} for
{course_name}.")
    except ValueError as e:
        print(e)
        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
```
This code segment demonstrates dictionary creation (student_data), input validation using the .isalpha() method to ensure names contain only letters, custom ValueError exceptions with descriptive messages, and a general Exception handler as a catch-all for unexpected errors.
I thoroughly tested the program in PyCharm, VS Code, and the terminal command line. The testing included both successful registrations and intentional error conditions: entering numbers in name fields, providing invalid file paths, entering non-numeric GPA values, and testing with corrupted JSON files. The program handled all scenarios correctly, displaying appropriate error messages without crashing.
My testing approach included:

- Valid inputs: proper names and course information
- Invalid name inputs: "John123" or "Smith456" to trigger ValueError
- Missing/corrupted JSON file to test FileNotFoundError
- Interrupted file operations to test the finally block

Conclusion

Module 5 proved to be one of the most challenging yet rewarding weeks in this course so far. The transition from working with simple lists to dictionaries, combined with JSON file handling and exception management, required a significant mental shift in how I approached Python programming. My biggest takeaway was the importance of proper code structure and indentation, something that seems minor but proved critical when implementing nested try/except blocks.
The repetitive practice through Labs 1-3, while frustrating at times with countless indent errors, ultimately built the muscle memory I needed to successfully complete Assignment 5. As someone who describes myself as a "110% type of person," I approached each failed attempt as an opportunity to better understand the mechanics of exception handling rather than just getting the code to work.
Moving forward, I'm more confident in my ability to write Python programs that handle errors gracefully instead of crashing when unexpected situations arise. The combination

of dictionaries for structured data, JSON for storage, and exceptions for error handling provides a solid foundation for building real-world applications, similar to the database systems I work with daily.
GitHub integration also reinforced the importance of version control and documentation, skills that translate directly to my professional work managing databases across teams statewide.