Melissa Garcia Ospina

Feb. 25, 2026

Foundations of Programming: Python

Assignment 05

GitHub Link

Advanced Collections and Error Handling

Introduction

This document explains how data structures, lists vs dictionaries, file input and output (read and write using JSON), error handing practices and collaboration tools, such as local vs remote repositories, are fundamental concepts needed to build dependable, maintainable and collaborative software.

Advanced Collections

Dictionary and List

One can think of a dictionary as a data structure that stores values in "keys". The Keys are case-sensitive, and one must use double-quote around them. This is not a requirement with Python, however, when working with JSON (JavaScript Object Notation) it does make a difference. When using dictionaries vs. lists while reading and writing data to files, one must keep in mind the following differences:

- With data organization, a list stores each line as an element in the list. However, for dictionary, one maps keys to values for each line in the file
- When accessing data, a list uses index numbers to obtain specific lines or elements, while in a dictionary, one can access a specific piece of data by using the associated key.
- In Data validation, lists do not provide a coherent way to ensure the correct indexing and data types are used. However, in a dictionary, since one uses keys, it is easier to check if the values of these keys are correct.
- While doing data transformation, a list may need addition coding to process each data within the line. While in a dictionary it is easier to simplify this process and access individual values for each key.

These are some main differences between a dictionary and a list, and it all comes down to the data that is being used and how one wants it to be processed.

JSON and CSV Files

JSON is a lightweight data-interchangeable format that is easy for humans to read and write and machine parse and generate (Mod05-Notes) (external link). It is used for communication between a

client and server, and for configuration files and data storage. In JSON files keys are strings in double quotes and values can be a variety of strings, numbers (can be integers or floats), objects {}, arrays [], Booleans or null. JSON gives the ability to nest objects and arrays within objects and arrays.

CSV files are a tabular format that is used for import/export, data storage when handling a large dataset. While JSON files are used to demonstrate structured data with different complexities, used for web APIs, document-oriented databases and configurations. Just like dictionaries and lists there are some differences depending on the data that is being used and the processing of it.

- In data structure, JSON uses key-value pairs and provides flexibility when representing complex and nested data structure. CSV on the other hand is a flat file that stores data in rows and columns and is not able to support nested structured with different hierarchy as JSON.
- As for complexity and usage, JSON is able to have interchangeable data between systems, such as within JavaScript. JSON is also suitable for NoSQL databases that store data in a document-oriented format. CSV is used for spreadsheets that export and import data to and from applications, an example is Microsoft Excel. It is also used for database backup and restoration given its simplicity.

Error Handling

When one is creating a code, it is helpful to anticipate that the user or other programmer may introduce new bugs when using the code. Error handling allows one to control errors and create a process of recovering from errors within the code. Many languages allow the use of try-except construct that allows to customize error messages. There is also the Exception class, which holds information about the error that just occurred when the code is running (Mod05-Notes) (external link). This can be used through the try-except block. In Python, it automatically creates an Exception object, as shown in Figure 1, where it fills with information of the error that caused the exception. The “e.__doc__)” provides a description of the exception type and its significance. The “e.__str__()” can include more details and the error message. There is also the “raise” errors, which is to give a warning to the user that their input needs to have specific requirements. There are many other error handling tools that one can use. The overall significance for error handling is to anticipate any errors that might occur.

```
try:
    student_first_name = input("Enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError("The first name must not be alphanumeric.")
    student_last_name = input("Enter the student's last name: ")
    if not student_last_name.isalpha():
        raise ValueError("The last name must not be alphanumeric.")
    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("Error: There is a problem with the entered data.")
    print("--Technical Error Message--")
    print(e.__doc__)
    print(e.__str__())
continue
```

Figure 1. Error Handling Example

Creating the code

I created a code that demonstrates the use of constants, variables and print statements to display a message about a student's registration for a Python course while implementing data processing using dictionaries and exception handling. I started with Importing the JSON file, defining data constants and variables. I then extracted the data from the file as shown in Figure 2.

```
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
except Exception as e:
    print ("Error:There was a problem reading the file. Please check file format.")
finally:
    if file is not None and file.closed == False:
        file.close()
```

Figure 2. Extracting Data from JSON File

Secondly, I did a loop to process the data implementing error handling as shown in Figure 3.

```
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 must not be alphanumeric.")
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("The last name must not be alphanumeric.")
            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("Error: There is a problem with the entered data.")
            print("--Technical Error Message--")
            print(e.__doc__)
            print(e.__str__())
        continue
```

Figure 3. Loop and Error Handling

Lastly, I saved the file while including the new JSON file and error handling, as shown in Figure 4.

```
# Save the data to a file
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")
        json.dump(students, file, indent=2)
        for student in students:
            print(f'Student {student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}')
    except Exception as e:
        print ("Error:There was a problem writing to the file.")
        print ("Please check the file is not already open.")
        print ("--Technical Error Message--")
        print(e.__doc__)
        print(e.__str__())
        continue
    finally:
        if file is not None and file.closed == False:
            file.close()
    continue
```

Figure 4. Saving to JSON file

Importing files to GitHub

Sharing codes foster collaboration and it can greatly improve efficiency in software development. Network file sharing is often used by developers when working on the same file. It is essentially a shared folder on a computer network. Cloud file sharing is similar to network file sharing, where it allows multiple users to collaborate on one file. However, it stores the data on remote servers hosted by cloud service providers. GitHub is one of the cloud-based programs that uses cloud-based file sharing elements, while focusing on code hosting and version control.

When getting into GitHub, I created an account and then created a repository (shared folders) where I was able to upload my files, as shown in Figure 5. It is helpful to know that any file that gets uploaded and be viewed, downloaded and modified by anyone that has the link.

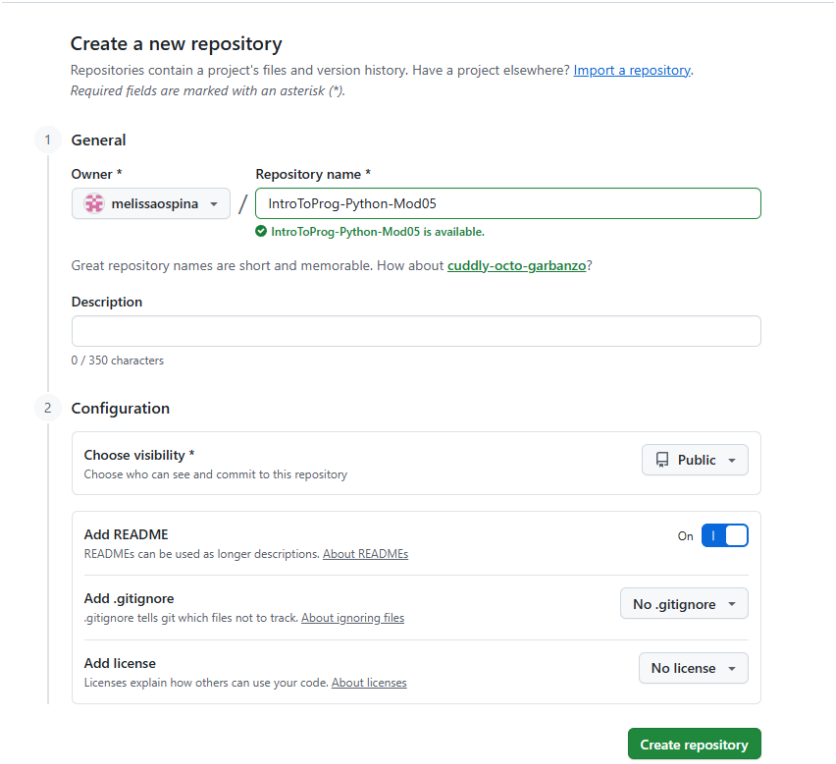

Figure 5. GitHub Repository

Summary

Understanding the following concepts makes programs more effective and shareable. Lists and dictionaries allow the use of different data structures. Pythons error handling allows one to catch and manage anticipated failures, prevents crashes and allows for a graceful recovery. Project files are stored on local systems or version-controlled repositories. GitHub is a platform that hosts Git repositories and adds collaboration features. Overall, all these elements – data structure, reliable file interface, error

handling and controlled collaboration – create a foundation for proper maintainability and collaborative software codes.