Shannon Brown

February 12, 2024

IT FDN 110 A

Assignment 5

Dictionaries and Exception Handling

Introduction

The focus of this week was utilizing dictionaries in parallel with lists, working with JSON files, and managing structured error handling (i.e., try-except). We reviewed video modules, readings, and labs on these topics. The purpose of this paper is to outline how we expand on our previous program from Assignment04, but with the addition of data processing using dictionaries and exception handling.

Creating a Python Script

The aim of Assignment 05 was to expand on script from Assignment 04 and create code with error handling to manage errors for outside users, while allowing users to select from a menu and accomplish the below goals:

(1) Prompt user to enter the student's first name, last name, and course name using input()
(2) Present a coma-separated string by formatting the collected data using print()
(3) Open and write the above collected data in a JSON file.
(4) End the program.

First, I opened Assignment05-Starter.py in PyCharm IDE, updated the header, and resaved the file as Assignment05.py. The starter file was pulled from Assignment04, and therefore, already contained many pre-written components, such as defined data constants. I went through a checked these constants and variables against the acceptance criteria. Since the aim of the assignment was to save my data as a JSON file, I changed the FILE_NAME constant to "Enrollments.json" (Figure 1).

```
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"

# Define the Data Variables and constants
student_first_name: str = ''  # Holds the first name of a student entered by the user.
student_last_name: str = ''  # Holds the last name of a student entered by the user.
course_name: str = ''  # Holds the name of a course entered by the user.
json_data: str = ''  # Required by acceptance criteria but not used
student_data: dict = {}  # one row of student data
students: list = []  # a table of student data
file = None  # Holds a reference to an opened file.
menu_choice: str  # Hold the choice made by the user.
```

Figure 1. Verify and adding data constants and variables needed for Assignment05

Similar to Assignment03, one main requirement was to start the program by automatically reading the file so the program doesn't write over pre-existing data, but adds to it. Before writing the required script to open and read the JSON file, I added the JSON module build-in associated with python at the top of my code (import json). Next, I used the open() function with the read mode to read the data. The json.load() function was used to parse the JSON data from the file into a list of dictionaries. I closed the file using the close() function to ensure the information was saved (Figure 2).

```
# When the program starts, extract the data from the file
    file = open(FILE_NAME, 'r')
    students = json.load(file)
    file.close()
```

Figure 2. Utilization open() and the json.load() functions to read and parse through the file data

As required by the acceptance criteria, I went back with try-except blocks to prevent errors later in code. For example, I set up exceptions to account for if the file isn't found using the FileNotFoundError. In addition, the "if file.closed == False" statement was employed to check whether the file was close properly and if not said code prevents an unclosed file (Figure 3).

```
# When the program starts, extract the data from the file
try:
    file = open(FILE_NAME, 'r')
    students = json.load(file)
    file.close()
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.closed == False:
        file.close()
```

Figure 3. Adding try-except block to the code used to extract data from the file

Next, I continued to review the pre-written code. The while loop code and the script for viewing the menu remained unchanged. In accordance with the acceptance criteria, for choice 1, while the input() functions remained the same, I changed the code to assign these inputs as a dictionary collection (student_data). Dictionaries use "Keys" to allow you to identify a value in a row by the column. For my code, the key names were FirstName, LastName, and CourseName. For the name inputs, I used the isalpha() function, which returns true if all the characters in the string are alphabetically, to provide an error code if the input didn't meet this criteria. Then, I used a try-except block, specifically with a ValueError, to account for any additional invalid entry mistakes (Figure 4).

```
# 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 a number")
        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain a number")
        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("User entered invalid information. Continuing...")
        continue
```

Figure 4. Creating a dictionary collection with inputted user data

The assignment required that upon selecting choice 3, a coma-separated string be printed. To complete this, I used a “for” loop function which allowed me to automatically loop over dictionary collection. The print() function was then used to produce an output based on previous user entries (Figure 5).

```
# Process the data to create and display a custom message
    print("-"*50)
    for student in students:
        print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
    print("-"*50)
    continue
```

Figure 5. Using a “for” loop to present a coma-separated string of dictionary data

Option 3’s purpose was to save data to the file. I opened the file “Enrollments.json” in write mode using the open() function. Using json.dump() function, I loaded the student data into the JSON file. I closed the file using the close() function to ensure the information was saved. I used try-except blocks, specifically with a TypeError and Exception, to account for any additional invalid data formatting issues or technical errors. In addition, to ensure the data file was closed properly, I used the if file.closed == False” statement (Figure 6). Like in Assignment 05, the final option (4) allowed the user to exit the program. A break statement was used to allow the user to exit the loop as its code.

```
    elif menu_choice == "3":
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file)
            file.close()
            continue
        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.closed == False:
                file.close()
```

Figure 6. Script used to open a JSON file, write in data, and close (i.e., save) the data to the file

Confirming Functionality of the Script

On PyCharm, I right clicked and then selected "Run Assignment05" to test my code throughout, and utilized debugging tools to fix any small errors, including missing syntax (e.g., quotes, parentheticals). Once the code was sufficiently tested on PyCharm, I opened the command prompt to test the program in the terminal. To use python to run my created program, I typed "python.exe" followed by the Assignment04.py, after relocating to the program's home directory. I confirmed the code was run correctly on the terminal, and met all parameters set by the acceptance criteria.

Adding Code to GitHub

Lastly, I logged into my GitHub account that I created with my UW email address. I created a repository called "IntroToProg-Python-Mod05" under my account (see https://github.com/sdbrown8/IntroToProg-Python-Mod05) and provided a short description (Figure 7). Then I uploaded both my files to the repository and committed my changes to save my work.

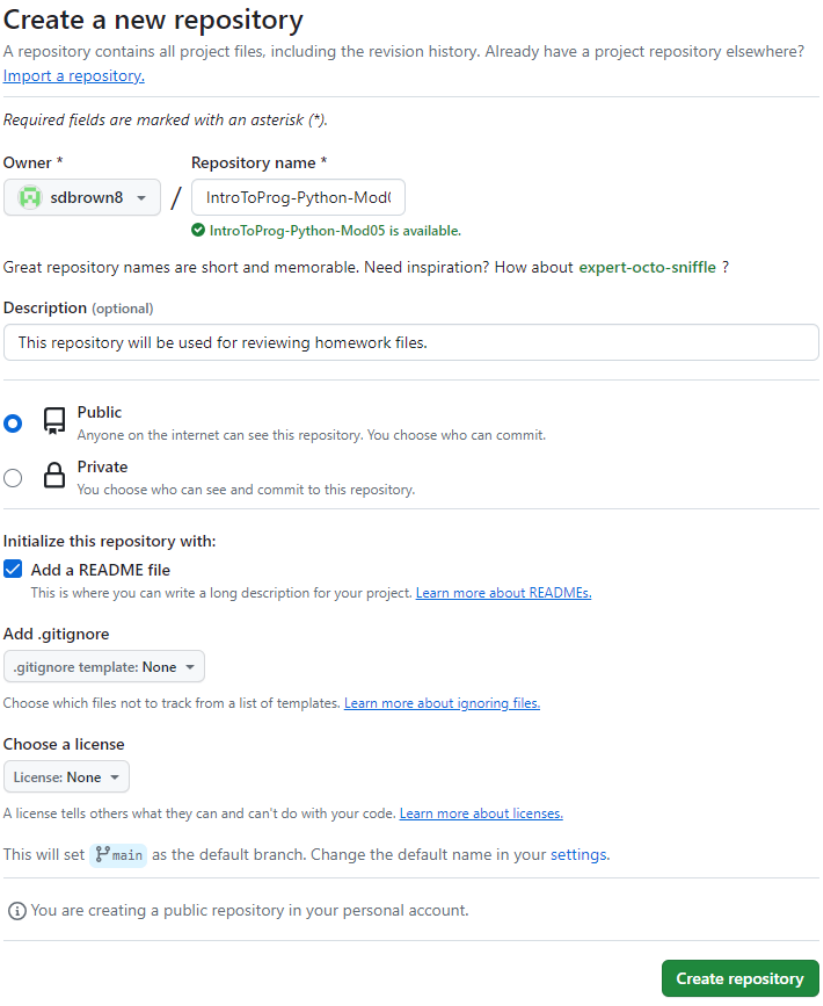

Figure 7. Creating a repository on GitHub

Summary

Assignment 05 relied on our previous knowledge of lists and working with files, but expanded on previous code to use dictionary collections, JSON files, and exception handling. All information required for the assignment was available via the module videos, readings, or labs. These tools will be further employed when I work with larger data sets in the future.