

Dictionaries & Error Handling

Introduction

This assignment builds on last week's assignment of creating a menu program. It allows users to register one or more students and displays the data in a table format, while storing the data in a dictionary, and also giving an option to save that data to a csv file. Throughout the program is error handling which enables the program to continue running if it runs into an error and displays a message to the user with user-friendly and technical error messages. The code along with this knowledge document is also hosted on my GitHub repository [here](#).

Creating the Script

Similar to the last assignments, I start my script off with a header, importing the exit function and defining the constants and variables. The constants stayed the same as last week, while the students_data variable changed to a dictionary. This change also updated my student variable to be a list of my dictionaries (Figure 1.1).

```
# Define the data variables
file = None
student: str = ""
students: list[dict[str, str]] = []
menu_choice: str = ""
student_first_name: str = ""
student_last_name: str = ""
course_name: str = ""
student_data: dict = {}
csv_data: str = ""
```

Figure 1.1: Updated students and student_data variables to include dictionaries.

The first thing the program does when it launches is read the contents of the enrollments.csv file (which was defined as the constant earlier) as a dictionary by indexing the line data and assigning the values to key". I also added error handling to display a message to the user if the file had not been created before selecting option one (Figure 1.2). I specified the FileNotFoundError directly as I'm using the with/as to open the file instead of open/close so the file will automatically close if it's found and opened.

```
# Open enrollments.csv and loads data to lists at program launch
# Error handling included if file not found
try:
    with open(FILE_NAME, "r") as file:
        file_data = file.readlines()
        for student in file_data:
            line = student.strip().split(",")
            row_dict = {
                "first_name": line[0],
                "last_name": line[1],
                "course_name": line[2]
            }
            students.append(row_dict)
        print("INFO: enrollments.csv file read into database.")
except FileNotFoundError:
    print("ERROR: Database file not found.")
```

Figure 1.2: Opening and reading the enrollments.csv file unless it's not found, in case it returns an error message.

My while loop remained the same from last week's assignment and presents the menu options to the user. Menu option 1 still prompts the user to enter student and course information; however I added error handling and instead of appending the data as a list to "student", it's appending as a dictionary row (Figure 1.3). I created specific ValueError for the blank or non-alphanumeric values for student_first_name and student_last_name and blank for course_name. The error message displayed to the user contains both my custom, user-friendly message along with the technical error details below that. I also included a line break with dashes to increase readability.

```
# Prompts the user for the registration information
if menu_choice == "1":
    # SError handling for non-alpha or empty strings
    try:
        student_first_name = input("Enter the student's first name: ")
        if len(student_first_name) == 0:
            raise ValueError("First name can't be empty.")
        if not student_first_name.isalpha():
            raise ValueError("First name can't contain non-alphanumeric values.")
        student_last_name = input("Enter the student's last name: ")
        if len(student_last_name) == 0:
            raise ValueError("Last name can't be empty.")
        if not student_last_name.isalpha():
            raise ValueError("Last name can't contain non-alphanumeric values.")
        course_name = input("Enter the course name: ")
        if len(course_name) == 0:
            raise ValueError("Course name can't be empty.")
        # Adds user input to dictionary
        student_data = {
            "first_name": student_first_name,
            "last_name": student_last_name,
            "course_name": course_name
        }
        students.append(student_data)
        # Prints confirmation that user's input has been accepted
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except Exception as error_details:
        print("-"*50)
        print(f"ERROR: {error_details}")
        print("***Technical Error Details**")
        print(error_details.__doc__)
```

Figure 1.3: Prompts for menu option 1 along with error handling and adding dictionary row(s) to data.

Menu option 2 still prints out the current data pulled in a loop out of "student". The change for this week is that my f-string pulls the values based on the dictionary key instead of by the list index.

```
# Prints out all users input in a csv format
elif menu_choice == "2":
    print("The current data is:")
    for student in students:
        print(f"{student['first_name']}, {student['last_name']}, {student['course_name']}")
```

Figure 1.4: For loop print statement to output all data from "students" based on dictionary key.

Menu option 3 slightly changed to use the dictionary keys when creating the “csv_data” variable instead of using the list index. I also added generic exception error handling in case the program runs into an error (Figure 1.5). If I had used open/close to write to the file instead of with/as, I could have added a condition to check if the file is closed and provided a specific error message for that. Since with/as automatically closes the file, I don’t believe it’s needed here.

```
# Adds user's input to csv file
elif menu_choice == "3":
    try:
        with open(FILE_NAME, "w") as file:
            for student in students:
                csv_data = f"{student['first_name']}, {student['last_name']}, {student['course_name']}\n"
                file.write(csv_data)
            print("INFO: New registrations have been saved to file.")
    except Exception as exception_details:
        print("-"*50)
        print(f"UNKNOWN ERROR: {exception_details}")
        print("---Technical Error Details---")
        print(exception_details.__doc__)
```

Figure 1.5: Appending user’s input to csv file by separating each value with a comma.

Menu option 4 and the catch all case remained the same from last week’s assignment. Option 4 closes the program, and the catch all lets the user know if they enter an invalid input (Figure 1.6).

```
# Exits the program
elif menu_choice == "4":
    print("Program Ended")
    exit()
# Displays a message if the user enters a value other than 1-4
else:
    print("That's not a valid option.")
```

Figure 1.6: Remaining menu options for exiting program or an invalid input.

Testing the Script

I tested my code in both VSCode’s terminal and IDLE. My testing followed the same steps as last week, except I checked that my error messages for the file not existing (Figure 2.1) and invalid options entered as part of menu option 1 worked as expected (Figure 2.2).

```
PS C:\Users\adavi\OneDrive\Documents\UW\FDN110 - Foundations of Python\Module05\Assignment> python assignment05.py
ERROR: Database file not found.

---- Course Registration Program ----
Select from the following menu:
1. Register a student for the course
2. Show current data
3. Save data to file
4. Exit the program
-----

Choose a menu option (1-4): █
```

Figure 2.1: File not found error message when launching program.

```
Choose a menu option (1-4): 1
Enter the student's first name:
-----
ERROR: First name can't be empty.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Alicia
Enter the student's last name:
-----
ERROR: Last name can't be empty.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Alicia
Enter the student's last name: Davis
Enter the course name:
-----
ERROR: Course name can't be empty.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

```
Choose a menu option (1-4): 1
Enter the student's first name: 2
-----
ERROR: First name can't contain non-alphanumeric values.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

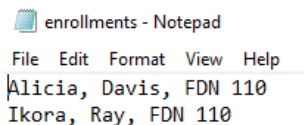
Figure 2.2: Error messages for blank and non-alphanumeric inputs for menu option 1.

Once valid data is entered, menu option 2 displays the current data (Figure 1.3).

```
Choose a menu option (1-4): 2
The current data is:
Alicia, Davis, FDN 110
Ikora, Ray, FDN 110
```

Figure 2.3: Current data in the “student” list.

Menu option 3, creates the enrollments.csv file and saves the new user input to the file (Figure 2.4).



```
enrollments - Notepad
File Edit Format View Help
Alicia, Davis, FDN 110
Ikora, Ray, FDN 110
```

Figure 2.4: CSV file with data from user input.

Any input outside 1-4 still results in an invalid option message (Figure 2.5), and option 4 still ends the program (Figure 2.6).

```
Choose a menu option (1-4): 5
That's not a valid option.
```

Figure 2.5: Invalid option

```
Choose a menu option (1-4): 4
Program Ended
```

Figure 2.6: Program Ended

Similar to in VSCode’s terminal, my testing followed the same steps in IDLE, except since the file exists from my testing in terminal, I don’t receive the error message that the file can’t be found. I do receive the correct error message for invalid inputs in menu option 1 though (Figure 2.7).

```
Choose a menu option (1-4): 1
Enter the student's first name:
-----
ERROR: First name can't be empty.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Fred
Enter the student's last name:
-----
ERROR: Last name can't be empty.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

```
Choose a menu option (1-4): 1
Enter the student's first name: Fred
Enter the student's last name: Flintstone
Enter the course name:
-----
ERROR: Course name can't be empty.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

```
Choose a menu option (1-4): 1
Enter the student's first name: 2
-----
ERROR: First name can't contain non-alphanumeric values.
**Technical Error Details**
Inappropriate argument value (of correct type).
```

Figure 2.7: Error messages for blank and non-alphanumeric inputs for menu option 1.

Once valid data is entered, I can see these users were added to my master list when selecting option 2 (Figure 2.8).

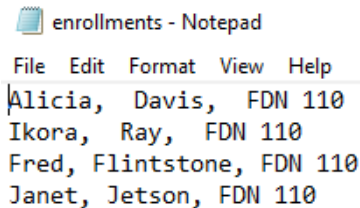
```
Choose a menu option (1-4): 2
The current data is:
Alicia, Davis, FDN 110
Ikora, Ray, FDN 110
Fred, Flintstone, FDN 110
Janet, Jetson, FDN 110
```

Figure 2.8: New users added to master list.

Option 3 saves the data as comma-delimited to the existing csv file (Figure 2.9 & Figure 2.10).

```
Choose a menu option (1-4): 3
INFO: New registrations have been saved to file.
```

Figure 2.9: Message to user that new input has been saved.



enrollments - Notepad

File Edit Format View Help

Alicia, Davis, FDN 110
Ikora, Ray, FDN 110
Fred, Flintstone, FDN 110
Janet, Jetson, FDN 110

Figure 2.10: Updated CSV File

Any input outside 1-4 still results in an invalid option message (Figure 2.11), and option 4 still ends the program (Figure 2.12).

```
Choose a menu option (1-4): 5
That's not a valid option.
```

Figure 2.11: Invalid option

```
Choose a menu option (1-4): 4
Program Ended
>>>
```

Figure 2.12: Program Ended

Summary

This week went smoother than last week in terms of testing. I changed my dictionary keys halfway through my code when I was trying to condense and was able to troubleshoot and solve it based on the error message I received. I also learned how to upload files to GitHub in the UI, but was unable to do so via the terminal.