George Nuesca

November 15, 2023

IT FDN 100 A

Assignment 05

Student Data with Dictionaries and Exceptions

Introduction

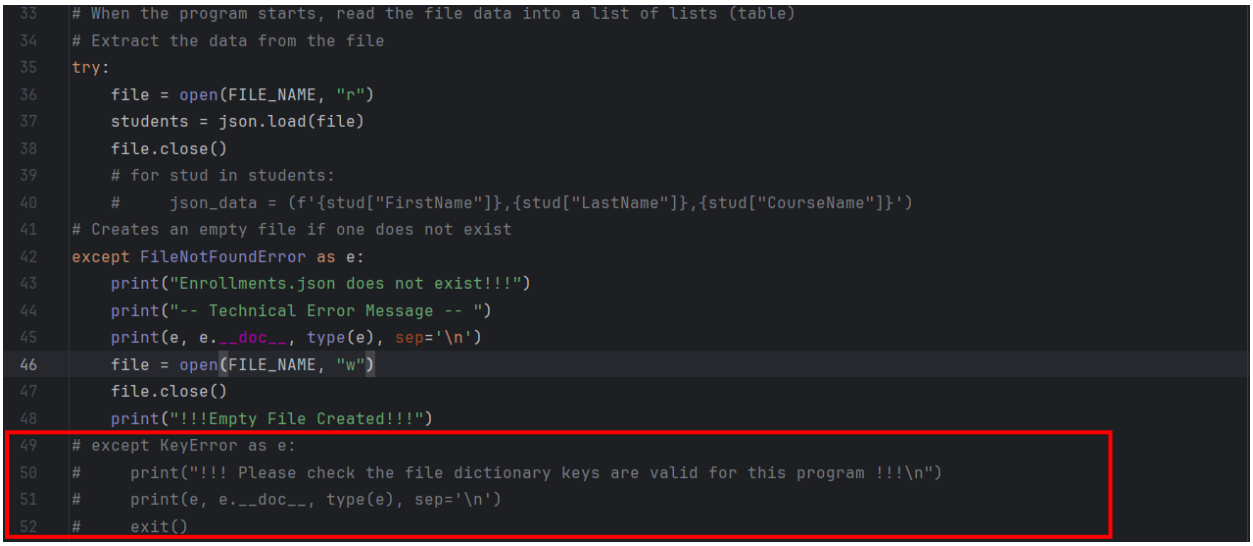
This paper documents how a menu program is constructed using dictionaries and handling errors. Having already completed the basic code of the menu, exception handling was added to allow for a missing file, request proper formatting of data entry, and identify if the file has been formatted correctly. Also different from the previous assignment, dictionaries replace lists and assist in handling errors.

Adding Code

Addressing File Formatting upon Initialization

The program criteria now asks to address file formatting issues. This is presented in two main issues, one with no existing file, and another with no or improper dictionary formatting. Note, a properly formatted dictionary with incorrect keys is still accepted, but a better way to avoid future issues in the program is discussed in a following subsection.

Upon starting the program, a try-except construct checks to see if a file exists. If it does not exist, a FileNotFoundError is thrown and recognized by an exception. This exception creates the specified file and displays a message to the user explaining the issue and how it is remedied. This code is shown in **Figure 1**, lines 42-48.



```
33  # When the program starts, read the file data into a list of lists (table)
34  # Extract the data from the file
35  try:
36      file = open(FILE_NAME, "r")
37      students = json.load(file)
38      file.close()
39      # for stud in students:
40      #     json_data = (f'{stud["FirstName"]},{stud["LastName"]},{stud["CourseName"]}')
41  # Creates an empty file if one does not exist
42  except FileNotFoundError as e:
43      print("Enrollments.json does not exist!!!")
44      print("-- Technical Error Message -- ")
45      print(e, e.__doc__, type(e), sep='\n')
46      file = open(FILE_NAME, "w")
47      file.close()
48      print("!!!Empty File Created!!!")
49  # except KeyError as e:
50  #     print("!!! Please check the file dictionary keys are valid for this program !!!\n")
51  #     print(e, e.__doc__, type(e), sep='\n')
52  #     exit()
```

Figure 1: Exception for handling missing file.

```
except Exception as e:
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
    print("!!!Check file formatting!!!\n")
    # exit()
```

Figure 2: Exception for handling incorrectly formatted JSON file.

The code also recognizes if there is an improperly existing JSON file. If the file is blank or has an improperly formatted dictionary, and the catch-all error handling Exception notifies the user. Note, it does not prevent the user from continuing the program. An exit() method was not used because the criteria did not direct to do so, although would be advised as shown on line 58 of **Figure 2**.

Addressing Incorrect Dictionary Keys

An interesting error to handle is if incorrect keys are in a properly formatted dictionary. Specifically, if the keys in the file do not match what will be requested in the program, it is likely preferred the user address the issue before working on data entry. The criteria requests this addressed later in the program, which is why the code is commented out.

In any case, code was experimented with to detect if required information is missing, as shown in the red box of commented code in **Figure 1**. Under this statement if the correct dictionary keys are not part of a student's data, the program notifies the user to check the file and exits the program by throwing a KeyError. Commented code to exit out of the program is also suggested as shown.

Handling Structures for Names

When entering inputs into the program, a standard naming structure is required of the user. Although modern names may allow for almost anything, this program does not allow numbers in a student's name. This is performed by a conditional statement using the isalpha() method raises a ValueError and is handled by an exception, shown in the red boxes of **Figure 3**.

Figure 3: Exception for handling incorrectly formatted JSON file.

```
    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        try:
            # Input the data
            noWhiteSpace = 'Field can not be blank. Please enter a valid first name.'
            student_first_name = input("What is the student's first name? ")
            if student_first_name.isspace():
                raise ValueError(noWhiteSpace)
            if not student_first_name.isalpha():
                raise ValueError("The first name should only contain letters.")
            student_last_name = input("What is the student's last name? ")
            if student_first_name.isspace():
                raise ValueError(noWhiteSpace)
            if not student_last_name.isalpha():
                raise ValueError("The last name should only contain letters.")
            course_name = input("Please enter the name of the course: ")
            if course_name.isspace():
                raise ValueError(noWhiteSpace)
        except ValueError as e:
            print(e)  # Prints the custom message
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
            print('Please try again')
            continue
```

Additionally, if the user attempts to enter an empty string, the code uses the isspace() method to again notify the user of a ValueError, which is demonstrated in the orange boxes of **Figure 3**. There are other ways to execute this feature, such as a conditional statement with an empty string or determining if there are not characters in the variable, but the isspace() method elegantly resolves this issue.

Saving to File Exceptions

As noted earlier, it would be unlikely a try-except method be applied to the saving file menu item if issues are addressed earlier. None-the-less, a condition to handle a KeyError is flagged if valid dictionary keys ('FirstName', 'LastName', and 'CourseName') are not found for a student, preventing a write to the file. It would be preferred if the program stop at the point so the user can repair the error but continues back to the menu for this exercise.

Testing the Program

There were several different ways the program was tested, thus requiring a few JSON file iterations to excite an error. Files with incorrect dictionary keys, a non-existing file, and files with no or incorrect data were tested to identify if issues were caught.

Upon running with incorrect dictionary keys, it was observed using a KeyError would more readily focus on the specified error. This would provide a more succinct message to the user for what to correct.

Summary

This document illustrates how to use exceptions to practically handle potential errors encountered in a program. Using a dictionary approach also provides an extra tool to address errors. Not mentioned in the previous sections, exception handling was used in item 2 to help the user understand if dictionary key errors are apparent.

A significant amount of testing was used to understand how the try-except approach can resolve program issues. Using different JSON file formatting assisted in determining how to properly use exceptions for a given error. Indeed, a significant amount of additional error handling can be applied beyond the assignment criteria to fix any permutation of problems.