August Belhumeur
11/13/2023
IT FDN 110
Assignment 05 - Document Knowledge
Github Link

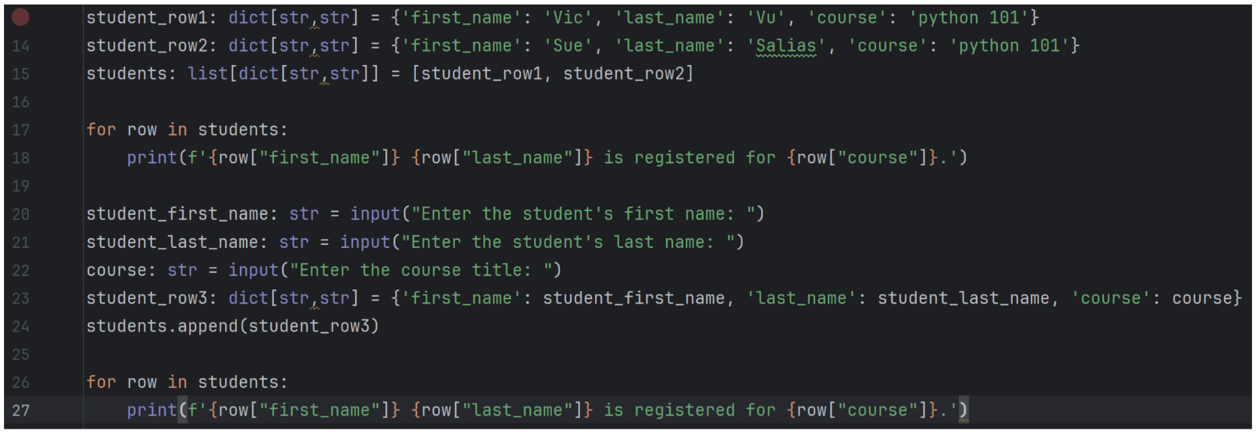
Try Dictionaries in JSON Files

Introduction

Module 05 introduced how to use dictionaries to store data values in “key:value” pairs and how dictionaries differ from other types of collections. We learned how to create dictionaries, isolate various parts of the dictionaries, and manipulate the data. We also learned how to adapt our code for JSON by importing the JSON library. This way, all data can be read and written to a JSON file as opposed to CSV files from previous modules. Lastly, we learned how to better address exception errors by using the try/except/else/finally statements to target a specific error and provide actions for the program to perform if that error comes up. This includes learning how to write our own exceptions to protect against other forms of incorrect inputs or outcomes. At the end of the module, we use Assignment 05 to showcase the use of dictionaries, JSON files, and structured error handling in a program that builds on Assignment 04.

Dictionaries

Dictionaries are collections that are ordered (as of Python 3.7), changeable, and without duplicates. They are written with curly brackets and used to store data values in “key:value” pairs similar to regular dictionaries that provide a term and the given term’s definition. Unlike lists and tuples, dictionaries do not use indices and instead use the key to uniquely identify a row in a table. Dictionaries can be stored within lists and, similar to lists, can contain user input values. **(See Figure A)**



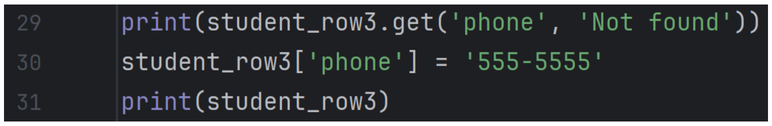
```python
student_row1: dict[str,str] = {'first_name': 'Vic', 'last_name': 'Vu', 'course': 'python 101'}
student_row2: dict[str,str] = {'first_name': 'Sue', 'last_name': 'Salias', 'course': 'python 101'}
students: list[dict[str,str]] = [student_row1, student_row2]

for row in students:
    print(f'{row["first_name"]} {row["last_name"]} is registered for {row["course"]}.')

student_first_name: str = input("Enter the student's first name: ")
student_last_name: str = input("Enter the student's last name: ")
course: str = input("Enter the course title: ")
student_row3: dict[str,str] = {'first_name': student_first_name, 'last_name': student_last_name, 'course': course}
students.append(student_row3)

for row in students:
    print(f'{row["first_name"]} {row["last_name"]} is registered for {row["course"]}.')
```

Figure A: Example code that showcases dictionary syntax.

August Belhumeur
11/13/2023
IT FDN 110
Assignment 05 - Document Knowledge
Github Link

Dictionaries have specific functions required to manipulate the data contained. You can add keys and values to an existing dictionary, use the .update method to change multiple values at a time, delete keys from a dictionary, pop keys from a dictionary, or print all keys, values, and items present. You can use “dictionary comprehension” to create a dictionary from a list or other dictionary in a single line of code. Similar to other collections of data, you can also export these sets of data from the program to an external file, such as a CSV or JSON file, and read data in a file back to the program. Python documentation includes additional functions for manipulating dictionary data.

If you try to print a key that doesn’t exist, you’ll get a KeyError. One solution for such an error is to use the .get method, which will search for a key in a dictionary and, if none are found, will return “None” instead of a KeyError that would end the program. You can even set custom text to return should a key not exist so that it prints something like “Not found” instead of “None”, or a more descriptive error message of your choosing. **(See Figure B)**



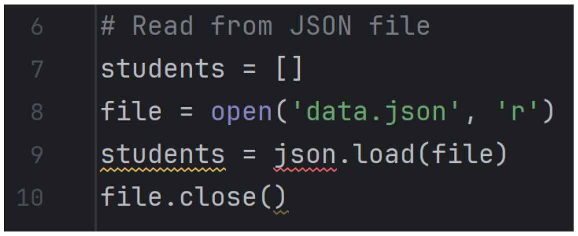
```
print(student_row3.get('phone', 'Not found'))
student_row3['phone'] = '555-5555'
print(student_row3)
```

Figure B: Example code for fixing a KeyError and then adding the missing key to the dictionary.

JSON Files

JSON files are popular to use across a lot of different programming languages and are structured very similarly to a list of dictionaries. JSON files and CSV files can hold the same information, but the JSON file is more verbose and includes more context for the objects in the list. While the same information can be added to a CSV file as a header to provide the additional context, the JSON file does this out of the box and a lot of libraries already have support for it.

To use JSON files, first adapt the code for JSON and import the JSON library at the top of the program by writing “import json”. Once the library is imported, you can write to and read from a JSON file just like a CSV, but the syntax to do so is shorter and more concise. To read from a JSON file, open the given file for reading and set the data you want to load from the file into the program as “json.load(file)”. Then, close the file. **(See Figure C)**

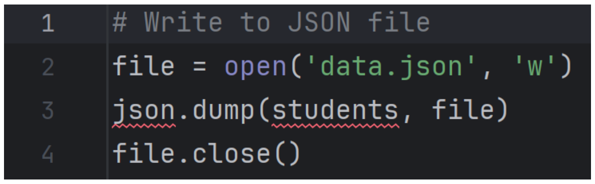


```
# Read from JSON file
students = []
file = open('data.json', 'r')
students = json.load(file)
file.close()
```

Figure C: Example code for how to read from a JSON file.

August Belhumeur
11/13/2023
IT FDN 110
Assignment 05 - Document Knowledge
Github Link

To write to a JSON file, open the given file for writing and use the “json.dump” statement to save data from the program into the specified file. Then, close the file. **(See Figure D)**



```python
# Write to JSON file
file = open('data.json', 'w')
json.dump(students, file)
file.close()
```

Figure D: Example code for how to write to a JSON file.

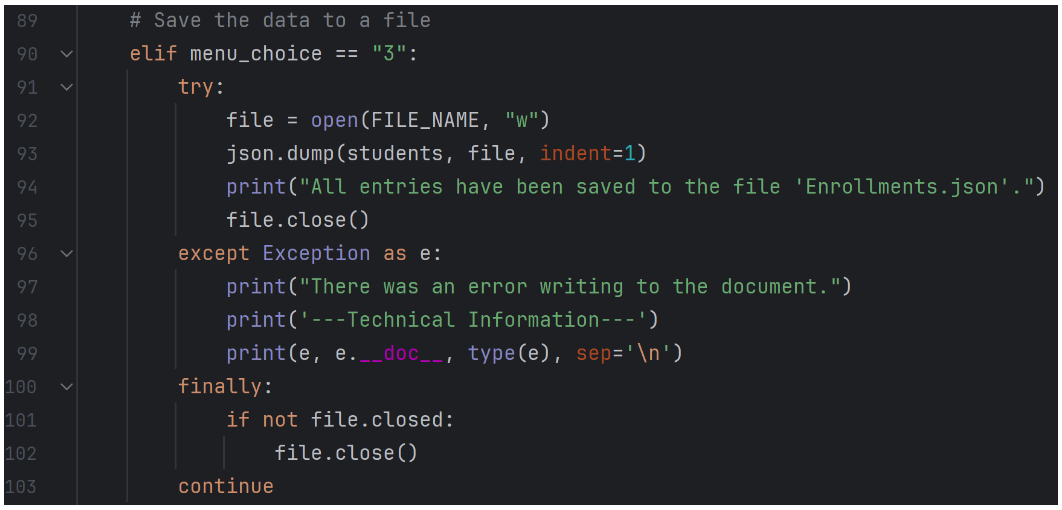
Try/Except/Else/Finally

In Python, two common types of errors are “syntax” errors and “exceptions”. A syntax error occurs when the parser detects an incorrect statement, such as having a missing bracket or incorrect punctuation for a phrase. Exception errors occur whenever Python code is syntactically correct, but still not functioning. To solve for exception errors, you can use “try”, “except”, “else”, and “finally” clauses. All of the code that you are trying to run would get wrapped in a section of code underneath a “try” statement that will execute up until the point where the first “except” is encountered. This is good to use for any code that might be kind of dangerous or have the potential to throw exceptions under different circumstances.

“Except” statements continue below “try” statements as if to say “try the code above, unless you encounter this particular error, in which case, do this other thing.” You can anticipate any number of exceptions underneath a try statement and differentiate how the program should respond should different errors arise. For example, if you are trying to open a file for reading, a FileNotFoundError may come up if that file has not yet been created. The exception could check to say if the file cannot be found, create a new file. You can also use exceptions to print technical information and documentation regarding the given error.

“Else” statements are used at the end of the list of exceptions to execute code in the absence of any exceptions. Did no exceptions come up when trying to run your original code? Then, run this code. Lastly, the “finally” clause appears at the end of all of these statements. This code will always get run and is an action to clean up after executing all of the try/ except / else code. For example, if you are opening a file for reading and running through exceptions related to this file, it’s a good idea to include code under “finally” to ensure that the file gets saved and closed before moving on to the rest of the program. **(See Figure E)**

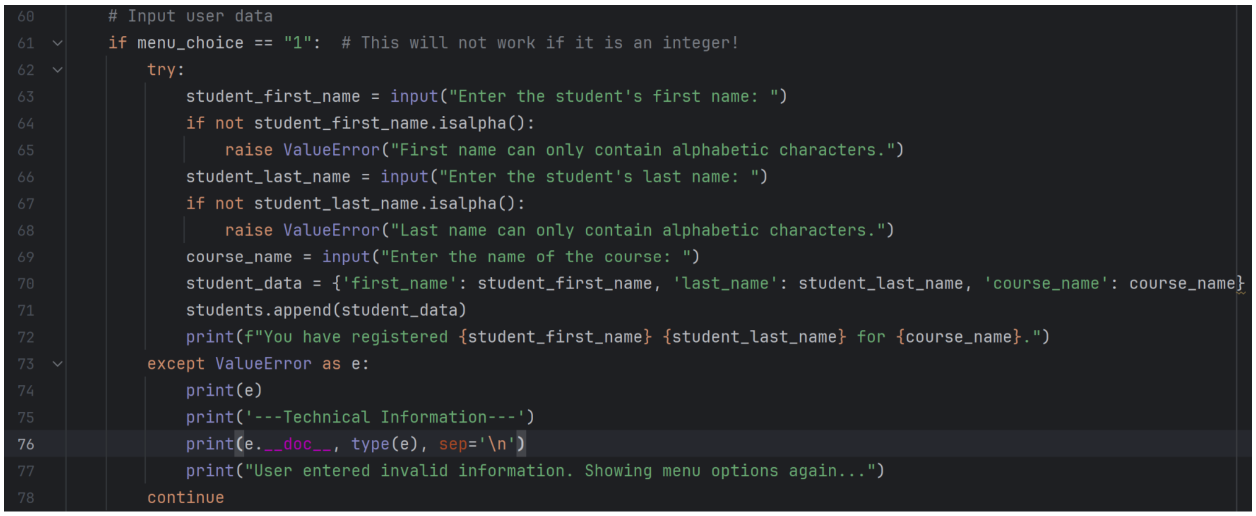
August Belhumeur
11/13/2023
IT FDN 110
Assignment 05 - Document Knowledge
Github Link



```
    # Save the data to a file
    elif menu_choice == "3":
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file, indent=1)
            print("All entries have been saved to the file 'Enrollments.json'.")
            file.close()
        except Exception as e:
            print("There was an error writing to the document.")
            print('---Technical Information---')
            print(e, e.__doc__, type(e), sep='\n')
        finally:
            if not file.closed:
                file.close()
        continue
```

Figure E: Example code from Assignment 05 of running try / except / finally to open a file for writing.

You can also add your own exceptions to a program in order to protect against incorrect information that Python wouldn't catch automatically, such as if a user were to input data incorrectly from what was instructed. These are called Value Errors and will throw a red error code exception and stop the program as if it were an error inherent to Python. However, if you don't want a Value Error to end the program, you can also place it under a try/except clause to address how the program should address the given error. **(See Figure F)**



```
    # 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("First name can only contain alphabetic characters.")
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("Last name can only contain alphabetic characters.")
            course_name = input("Enter the name of the course: ")
            student_data = {'first_name': student_first_name, 'last_name': student_last_name, 'course_name': 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 Information---')
            print(e.__doc__, type(e), sep='\n')
            print("User entered invalid information. Showing menu options again...")
        continue
```

Figure F: Example code from Assignment 05 of using Value Errors to protect against users inputting invalid data.

August Belhumeur
11/13/2023
IT FDN 110
Assignment 05 - Document Knowledge
Github Link

Creating the Script for Assignment 05

I began Assignment 05 by opening the starter file and copying the code into a new file. I edited the header for the new program and added my changelog information. I kept the data constants the same, but edited the data variables to remove the unused “csv_data” variable and set “student_data” to be of the type dictionary instead of list. Next, I added code to import the json library at the start of the program. On start, the program needs to read the contents from a JSON file titled “Enrollments.json”. The starter file has a section of code for reading data from a CSV file on start, so I converted the formatting of this code to read from a JSON file instead and added a try / except / finally clause to handle a FileNotFoundError that I encountered immediately upon testing the program given that I hadn’t created the JSON file yet. This except clause creates a blank JSON file of the correct name if such a file cannot be found and then proceeds to “finally” to ensure the file gets closed. If the file already exists, then the “try” clause will print the JSON file’s data to the program and then close the file. **(See Figure G)**

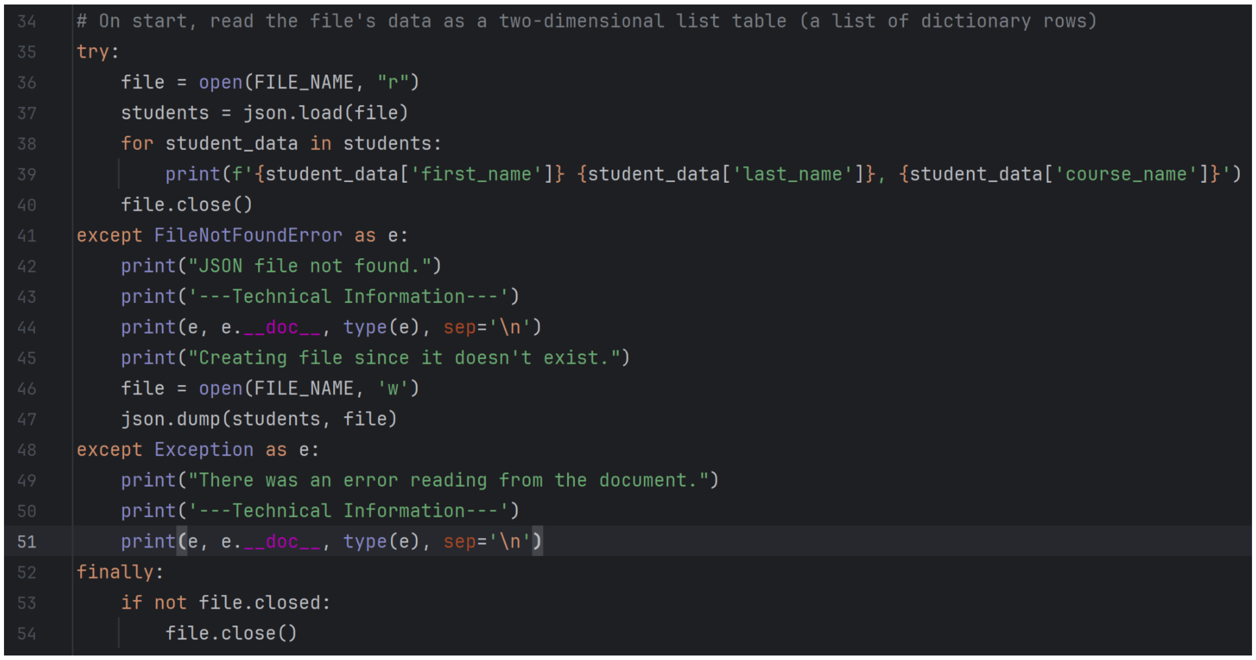
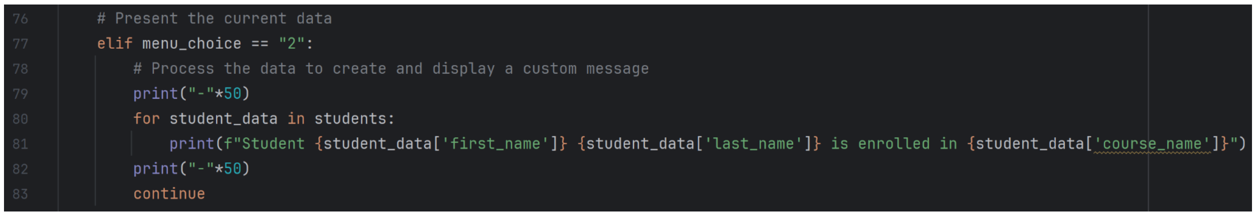

```python
# On start, read the file's data as a two-dimensional list table (a list of dictionary rows)
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    for student_data in students:
        print(f'{student_data['first_name']} {student_data['last_name']}, {student_data['course_name']}')
    file.close()
except FileNotFoundError as e:
    print("JSON file not found.")
    print('---Technical Information---')
    print(e, e.__doc__, type(e), sep='\n')
    print("Creating file since it doesn't exist.")
    file = open(FILE_NAME, 'w')
    json.dump(students, file)
except Exception as e:
    print("There was an error reading from the document.")
    print('---Technical Information---')
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if not file.closed:
        file.close()
```

Figure G: Code from Assignment 05 reading a JSON file’s data as a two-dimensional list table (a list of dictionary rows) at the start of the program.

Next, I edited the menu choices. For menu choice 1, the inputs largely stay the same, but I added a try/except clause to handle additional ValueErrors in order to protect against invalid user inputs. For two of the three user inputs (being “student_first_name” and “student_last_name”), the program should only accept alphabetical entries since a name should not include numerical values. The input variable “course_name” does accept numerical inputs, such as for a course called “Python 101”. These input values all get added to the dictionary “student_data” and appended to the list “students”. If the user does input invalid information, the

August Belhumeur
11/13/2023
IT FDN 110
Assignment 05 - Document Knowledge
Github Link
program will present the menu choices again and the user can select menu choice 1 to try to enter the student's data correctly. **(See Figure F)**

Menu choice 2 is mostly the same, except that the "print" statement had to be edited for correctly displaying dictionary data from the variable "student_data" in "students". **(See Figure H)**



```python
    # Present the current data
    elif menu_choice == "2":
        # Process the data to create and display a custom message
        print("-"*50)
        for student_data in students:
            print(f"Student {student_data['first_name']} {student_data['last_name']} is enrolled in {student_data['course_name']}")
        print("-"*50)
        continue
```

Figure H: Code from Assignment 05 presenting all current data to the user.

Menu choice 3 saves all of the input data to the JSON file. The starter file has code for this section that saves data to a CSV file, so I edited the format to save to a JSON file instead and included a try/except/finally clause in the case of any errors. In this clause, I'm not trying to catch any specific errors, but rather allowing any potential error to be displayed to the user with documentation as part of the program instead of crashing it. I also added the function "indent=1" to the json.dump statement in order to save all of the data to the json file on separate lines instead of one horizontal line. This makes the data easier to parse as side-scrolling is more cumbersome and visually louder than scrolling through the data vertically. **(See Figure E)** The remaining code from here to the end of the script has been kept the same from the starter file to the final submission, which includes menu choice 4 allowing the user to break out of the while loop, an else statement to account for the user selecting a menu option that is not valid, and a final print statement to tell the user that the program has ended when they break out of the loop.

Summary

In Module 05, we began using dictionaries to store collections of data in "key:value" pairs and learned ways to manipulate the data pairs. We learned how to read and write collections of data to JSON files, as opposed to CSV files, and how to address exception errors for this code by providing actions for the code to take should errors arise. In Assignment 05, we put these lessons to use by editing the program from Assignment 04 in order to showcase what we learned about dictionaries and JSON files. The program still allows the user to input, display, and save multiple registrations for students, but data now gets stored in dictionaries and lists of dictionaries. This data gets read from and written to a JSON file instead of a CSV. We improved the program further by adding structured error handling for reading and writing to a file and by creating ValueErrors to protect against incorrect user inputs.