

John D. Blake

15 May 2024

IT FDN 110 A Sp 24: Foundations Of Programming: Python

Assignment05

Dictionaries, JSON and Error Handling

Introduction

In Module 5, we covered some new ground:¹

- Dictionaries
- JSON
- Error Handling

These concepts were tied together in Assignment05, where we used them to further develop a script to read, input and save data. The following report expands on these objectives.

Dictionaries

Dictionaries are similar to lists, in that they used to store multiple data elements within the same entity. A major difference is in the 'Key: Value' structure of the dictionary entry. The syntax is as follows:

```
{"Key1": "Value1", "Key2":, "Value1", ... "KeyN": "Value1"}  
{"Key1": "Value2", "Key2":, "Value2", ... "KeyN": "Value2"}
```

Recall that in a list of length 'n' the indexing goes from 0 to n-1. In a dictionary, the indexing references the 'Key.' Thus, it can be a more intuitive way to reference items within the entity.

For example,

For list = [cow, milk, cheese], the reference [1] returns 'milk.'

For dictionary = {"Animal": "cow", "Raw_Material": "milk", "End_Product": "cheese"}, the reference "Raw_Material" returns 'milk.'

¹ We were also introduced to Github, which basically required signing up for and configuring an on-line account to act as a repository for code. I don't include that portion of the module in the write-up, as I didn't really learn much, apart from Github's existence and utility.

JSON Files

JSON (JavaScript Object Notation) is a portable data convention that is similar to the dictionary in that it uses a 'Key:Value' structure, and can store multiple rows of information. The basic syntax is as follows (for an $i \times j$ array):

```
[
  {"Key1": "Value1", "Key2":, "Value1", ... "Keyi": "Value1"},
  {"Key1": "Value2", "Key2":, "Value2", ... "Keyi": "Value2"},
  .
  .
  {"Key1": "Valuej", "Key2":, "Valuej", ... "Keyi": "Valuej"}
]
```

Error Handling

One of the challenges in working with computers is that when things go wrong, it can be difficult for the non-expert to understand the issue. This is often because while the accompanying error message is perfectly precise and accurate, it seems to have been written in riddles using Tolkienesque Elfish runes.

Another issue is that often when the code hits an error, it can just stop working, thus forcing the user to completely rerun the program.

Fortunately there is a cool feature within Python that allows the developer to add some 'user friendliness' to the routine by crafting a clear(er) message in response to a foreseen type of user error, as well as to allow the routine to continue so the user can respond appropriately.

The basic syntax of this is as follows:

```
try:
    #Runs first
    <code>
except:
    #Runs if exception occurs in the 'try' block
    <code>
else:
    #Runs if the 'try' block succeeds
    <code>
finally:
    #Always runs
    <code>
```

Assignment 05

We began with a common script, which was a carry-over from the previous module. We then had instruction to define and modify some of the constants and variables (see the script in Appendix 1). I'll go into some interesting features of this script:

Import modules - As we working with JSON files, we had to import the 'json' module. We also had to import a feature that would allow us to close a file when working through the Error Handling routines.²

```
import json #Import ... json stuff
import io as _io #Has something to do with closing file after Error handling
```

Read .json file - Here we read data in from an existing .json file. (Note that I had to clear out the 'students' list for the script to run reliably.³)

We include an error handling feature that anticipates a missing, misplaced, or misidentified filename, as well as a general exception message.

```
try:
    students = [] #THIS STEP IS SOMEHOW VERY SUPER-DUPER IMPORTANT!!!
    file = open(FILE_NAME, "r")
    students = json.load(file) #Reads json data into list
    file.close()
except FileNotFoundError as e:
    print("Text File Must Exist\n")
    print('In ... "Technical" Terms')
    print(e,e.__doc__, type(e), sep='\n')
except Exception as e:
    print("Non-specific error\n")
    print('In "Technical" Terms')
    print(e,e.__doc__, type(e), sep='\n')
finally:
    if file.closed==False:
        file.close()
```

Keyboard input - In this step, we use largely recycled code to prompt the user to input name and course information. We then format this as a dictionary entry and write the results to the screen using the appropriate 'key' reference. Additionally, we add an error handling routine in anticipation that the user inputs a number in a name entry.

```
# Step 1: Input user data
if menu_choice == "1": # This will not work if it is an integer!
    try:
        first_name:str = input("Enter the student's first name: ")
        if not first_name.isalpha():
            raise ValueError("First name should contain only letters.")
        last_name:str = input("Enter the student's last name: ")
        if not last_name.isalpha():
            raise ValueError("Last name should contain only letters.")
        course_name:str = input("Please enter the name of the course: ")
        student = {'FirstName': first_name, 'LastName': last_name, 'CourseName': course_name}
        students.append(student)
        print(f"You have registered {first_name} {last_name} for {course_name}.")
```

² I'm not 100% sure what this does, but it seems to make the script work.

³ See footnote #2.

```

except ValueError as e:
    print(e)
    print('In ... "Technical" Terms')
    print(e, e.__doc__)
    print(e.__str__())
except Exception as e:
    print("Non-specific error\n")
    print('In "Technical" Terms')
    print(e, e.__doc__, type(e), sep='\n')
finally:
    continue

```

Data Display - Again, we tweak existing code to print the data currently stored in the 'students' list, which is in dictionary format.

```

# Step 2: Present the current data
elif menu_choice == "2":
    # Process the data to create and display a custom message
    print("-"*50)
    for student in students:
        message = "{} {} is enrolled in {}".format(
            student["FirstName"], student["LastName"], student["CourseName"])
        print(message)
    print("-"*50)
    continue

```

Write to .json file - In this step, we use the '.dump' feature from the json module to write the data in the 'students' list to a .json file. Again, we add error handling in anticipation of an incorrect data formatting error.

```

# Step 3: Save the data to a file
elif menu_choice == "3":
    file = open(FILE_NAME, "w")
    try:
        json.dump(students, file) # 'dump' (write) to file
        file.close()
        print('The following data was saved to', FILE_NAME)
    except TypeError as e:
        print("Please check that the data is a valid JSON format\n")
        print('In "Technical" Terms')
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
        print("Non-specific error\n")
        print('In "Technical" Terms')
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if file.closed == False:
            file.close()
        continue

```

Finally, the routine to exit the program is completely reused from a previous assignment.

```

# Step 4: Stop the loop
elif menu_choice == "4":
    break # out of the loop
else:
    print("Please only choose option 1-4")

```

Verification

I developed and ran the script in PyCharm (see verification in Appendix 2), and I then ran the script in the macOS shell (see Appendix 3). In both instances, the script ran per the assignment's acceptance criteria.

Conclusion

The big takeaway from this lesson was the great simplification in file reading and writing operations with the built-in commands within the .json module. I imagine there are many such modules that we can use to further simplify routine operations.

The other interesting aspect is the error handling, as this requires the developer to anticipate typical pitfalls to make the recovery easier for the user.

Appendix 1: Python script

```
# ----- #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception handling
# Change Log: (Who, When, What)
#   RRoot, 1/1/2030, Created Script
#   JD Blake, 5/10/2024, CONSTANT and variable defn
#   JD Blake, 5/11/2024, (0) Read .json file, (1) Input, (2) Display and (3) Write .json file
#   JD Blake, 5/12/2024, Added Error Handling for (0) Read .json file and (1) Input
#   JD Blake, 5/13/2024, Added Error Handling for (3) Write .json file. Clean up draft.
# ----- #
# Define the Data Constants
MENU: str = '''
----- Course Registration Program ----
Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----
'''

# Define the Data Constants
FILE_NAME: str = ("Enrollments.json")
#FILE_NAME: str = ("Enrollment.json") #Misspelled Filename for FileNotFoung test
#FILE_NAME: str = ("DEBUG.json") #Empty data file for Exception Test

import json #Import ... json stuff
import io as _io #Has something to do with closing file after Error handling

# Define the Data Variables and constants
first_name: str = '' # Holds the first name of a student entered by the user.
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.
student: dict = {} # one row of student data
students: list = [] # a table of student data
json_data: str = '' # Holds string for json data
file = _io.TextIOWrapper # Holds a reference to an opened file.
menu_choice: str # Hold the choice made by the user.

# When the program starts, read the file data into a list of lists (table)
# Step 0: Extract the data from the file
try:
    students = [] #THIS STEP IS SOMEHOW VERY SUPER-DUPER IMPORTANT!!!
    file = open(FILE_NAME, "r")
    students = json.load(file) #Reads json data into list
    file.close()
except FileNotFoundError as e:
    print("Text File Must Exist\n")
    print('In ... "Technical" Terms')
    print(e,e.__doc__, type(e), sep='\n')
except Exception as e:
    print("Non-specific error\n")
    print('In "Technical" Terms')
    print(e,e.__doc__, type(e), sep='\n')
finally:
    if file.closed==False:
        file.close()

while (True):
    # Present the menu of choices
    print(MENU)
    menu_choice = input("What would you like to do: ")
```

```

# Step 1: Input user data
if menu_choice == "1": # This will not work if it is an integer!
    try:
        first_name:str = input("Enter the student's first name: ")
        if not first_name.isalpha():
            raise ValueError("First name should contain only letters.")
        last_name:str = input("Enter the student's last name: ")
        if not last_name.isalpha():
            raise ValueError("Last name should contain only letters.")
        course_name:str = input("Please enter the name of the course: ")
        student = {'FirstName': first_name, 'LastName': last_name, 'CourseName': course_name}
        students.append(student)
        print(f"You have registered {first_name} {last_name} for {course_name}.")
    except ValueError as e:
        print(e)
        print('In ... "Technical" Terms')
        print(e, e.__doc__)
        print(e.__str__())
    except Exception as e:
        print("Non-specific error\n")
        print('In "Technical" Terms')
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        continue

# Step 2: Present the current data
elif menu_choice == "2":
    # Process the data to create and display a custom message
    print("-"*50)
    for student in students:
        message = "{} {} is enrolled in {}".format(student["FirstName"], student["LastName"], student["CourseName"])
        print(message)
    print("-"*50)
    continue

# Step 3: Save the data to a file
elif menu_choice == "3":
    file = open(FILE_NAME, "w")
    try:
        json.dump(students, file) # 'dump' (write) to file
        file.close()
        print('The following data was saved to', FILE_NAME)
    except TypeError as e:
        print("Please check that the data is a valid JSON format\n")
        print('In "Technical" Terms')
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
        print("Non-specific error\n")
        print('In "Technical" Terms')
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if file.closed == False:
            file.close()
        continue

# Step 4: Stop the loop
elif menu_choice == "4":
    break # out of the loop
else:
    print("Please only choose option 1-4")

print("Program Ended")

```

Appendix 2 - Verification in PyCharm

Single Entry

```
/usr/local/bin/python3.12 /Users/johnblake/Desktop/Python_Class/work/A05/Assignment05/Assignment05.py
```

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

```
What would you like to do: 2
```

```
First_Name Last_Name is enrolled in Course_Name
```

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

```
What would you like to do: 1
Enter the student's first name: Tom
Enter the student's last name: Hanks
Please enter the name of the course: Drama 101
You have registered Tom Hanks for Drama 101.
```

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

```
What would you like to do: 2
```

```
First_Name Last_Name is enrolled in Course_Name
Tom Hanks is enrolled in Drama 101
```

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

```
What would you like to do: 3
The following data was saved to Enrollments.json
```

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

Enrollments.json data

```
{{"FirstName": "First_Name", "LastName": "Last_Name", "CourseName": "Course_Name"},
{"FirstName": "Tom", "LastName": "Hanks", "CourseName": "Drama 101"}}
```

Multiple Entries

---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 3

The following data was saved to Enrollments.json

---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 1

Enter the student's first name: Billy

Enter the student's last name: Strings

Please enter the name of the course: Bluegrass 101

You have registered Billy Strings for Bluegrass 101.

---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 1

Enter the student's first name: Adam

Enter the student's last name: Smith

Please enter the name of the course: Econ 101

You have registered Adam Smith for Econ 101.

---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 2

First_Name Last_Name is enrolled in Course_Name

Tom Hanks is enrolled in Drama 101

Billy Strings is enrolled in Bluegrass 101

Adam Smith is enrolled in Econ 101

---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 3

The following data was saved to Enrollments.json

---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course.
2. Show current data.
3. Save data to a file.
4. Exit the program.

What would you like to do: 4

Enrollments.json data

```
{ "FirstName": "First_Name", "LastName": "Last_Name", "CourseName": "Course_Name"},
{ "FirstName": "Tom", "LastName": "Hanks", "CourseName": "Drama 101"},
{ "FirstName": "Billy", "LastName": "Strings", "CourseName": "Bluegrass 101"},
{ "FirstName": "Adam", "LastName": "Smith", "CourseName": "Econ 101"}
```

Error Handling Examples

Here, I substitute a misspelled filename 'Enrollment.json' for the correct filename 'Enrollments.json.'

```
#FILE_NAME: str = ("Enrollments.json")
FILE_NAME: str = ("Enrollment.json") #Misspelled Filename for FileNotFoundError test

/usr/local/bin/python3.12 /Users/johnblake/Desktop/Python_Class/work/A05/Assignment05/Assignment05.py
Text File Must Exist

In ... "Technical" Terms
[Errno 2] No such file or directory: 'Enrollment.json'
File not found.
<class 'FileNotFoundError'>

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

What would you like to do:
```

Next, I use a number in a name input.

```
/usr/local/bin/python3.12 /Users/johnblake/Desktop/Python_Class/work/A05/Assignment05/Assignment05.py
```

```
---- Course Registration Program ----
```

```
Select from the following menu:
```

1. Register a Student for a Course.
 2. Show current data.
 3. Save data to a file.
 4. Exit the program.
- ```

```

```
What would you like to do: 1
```

```
Enter the student's first name: Jam3s
```

```
First name should contain only letters.
```

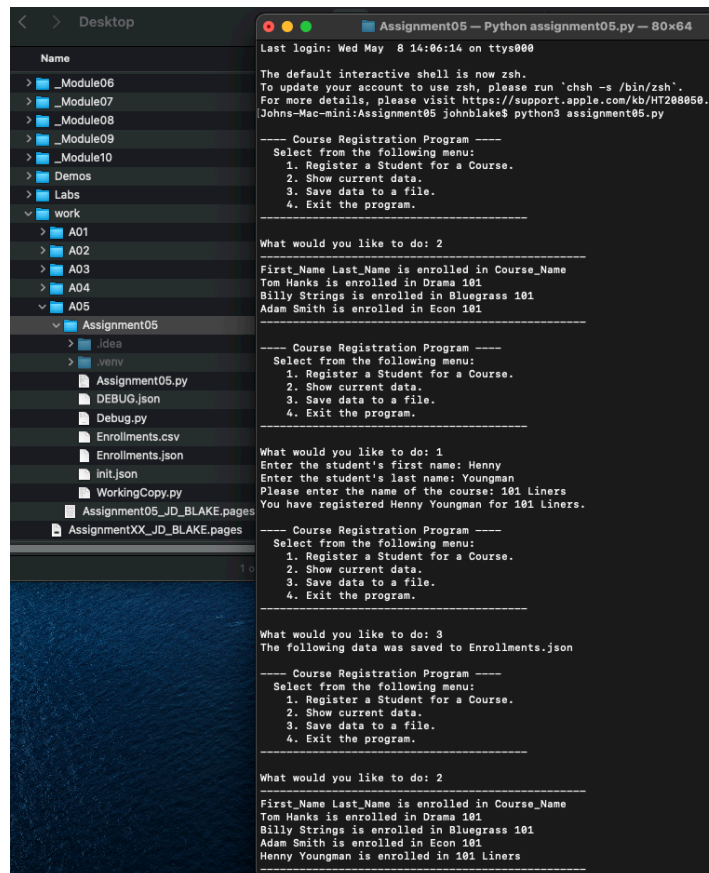
```
In ... "Technical" Terms
```

```
First name should contain only letters. Inappropriate argument value (of correct type).
```

```
First name should contain only letters.
```

## Appendix 3 - Verifying script runs in macOS

Here we see that the script reads the existing file, accepts new input, and writes to the .json file while in macOS.



```

Desktop
Name
 > _Module06
 > _Module07
 > _Module08
 > _Module09
 > _Module10
 > Demos
 > Labs
 > work
 > A01
 > A02
 > A03
 > A04
 > A05
 > Assignment05
 > idea
 > .venv
 Assignment05.py
 DEBUG.json
 Debug.py
 Enrollments.csv
 Enrollments.json
 init.json
 WorkingCopy.py
 Assignment05_ID_BLAKE.pages
 AssignmentXX_ID_BLAKE.pages

Assignment05 - Python assignment05.py - 80x64
Last login: Wed May 8 14:06:14 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run 'chsh -s /bin/zsh'.
For more details, please visit https://support.apple.com/kb/HT208050.
Johns-Mac-mini:Assignment05 johnblake$ python3 assignment05.py

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

What would you like to do: 2

First_Name Last_Name is enrolled in Course_Name
Tom Hanks is enrolled in Drama 101
Billy Strings is enrolled in Bluegrass 101
Adam Smith is enrolled in Econ 101

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

What would you like to do: 1
Enter the student's first name: Henny
Enter the student's last name: Youngman
Please enter the name of the course: 101 Liners
You have registered Henny Youngman for 101 Liners.

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

What would you like to do: 3
The following data was saved to Enrollments.json

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

What would you like to do: 2

First_Name Last_Name is enrolled in Course_Name
Tom Hanks is enrolled in Drama 101
Billy Strings is enrolled in Bluegrass 101
Adam Smith is enrolled in Econ 101
Henny Youngman is enrolled in 101 Liners

```