Chantal Van den bussche

2/26/2025

Foundations of Programming: Python

Assignment05

GitHub link:

Dictionaries, JSON files, and Error Handling

Introduction

This week, I learned about the difference between dictionaries and lists in Python and how to create and append dictionaries in Python. I also learned how to create, read, and write JSON files. In addition to learning about JSON files and dictionaries, I also learned how to build Error Handling within my program.

Dictionaries

While lists and dictionaries are fairly similar within Python in that they can store multiple variables within themselves, there are some key differences. Dictionaries are enclosed within curly brackets ({}) and are useful for when data has specific key-value pairings, such as data with named attributes. This is because Python dictionaries map specific keys to corresponding values. This has the added value of also making it easy to understand the data structure. As seen in Example 1, for my Assignment05, I mapped the key **“FirstName”** to the variable **student_first_name**, **“LastName”** to the variable **student_last_name**, and **“CourseName”** to the variable **course_name**. This mapping also makes it easy to access specific pieces of data using the associated keys [Example 2].

Example 1:

```
student_data = {"FirstName": student_first_name,
                "LastName": student_last_name,
                "CourseName": course_name}
```

Example 2:

```
print(f"Student {student["FirstName"]} "
      f"{student["LastName"]} is enrolled "
      f"in {student["CourseName"]}.")
```

JSON files

JSON stands for JavaScript Object Notation and is a file-type that is easy for both humans and machines to understand. JSON files are similar to Python dictionaries in that they consist of key-value pairs. JSON values can be string, number, object, an array, boolean, or null; and, JSON keys must be enclosed in double quotation (“”) marks. Similar to Python dictionaries, commas (,) separate JSON key-value pairs and dictionaries from each other as seen in Figure 1.

```
[{"FirstName": "Bob", "LastName": "Smith", "CourseName": "Python 100"}, {"FirstName": "Sue", "LastName": "Jones",
"CourseName": "Python 100"}, {"FirstName": "Chantal", "LastName": "Van den bussche", "CourseName": "Python 100"},
{"FirstName": "David", "LastName": "Nguyen", "CourseName": "Python 100"}, {"FirstName": "Chris", "LastName": "Kennedy",
"CourseName": "Python 100"}]
```

Figure 1: Commas separating JSON key-value pairs and dictionaries

JSON Module

Python has a built-in module that works with JSON data. This makes it very easy to read, import, append, and write JSON data using Python. For my Assignment05, I imported json into my program at the beginning of my code, before I even defined my constants and variables [Example 3]. I then used the json.load(file) method to parse the JSON data in my “Enrollments.json” file and store it in my list **students** [Example 4].

Example 3:

```
#
------------------------------------------------------------------------------
----------- #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception
handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Chantal Van den bussche,2/26/2025,Created & edited script
#
------------------------------------------------------------------------------
----------- #


# Import JSON module
import json


# 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.
----------------------------------------
'''
```

Example 4:

```
file = open(FILE_NAME, "r")
students = json.load(file)
file.close()
```

Later, in the main body of my code, I used the json.dump() method to easily write the contents of my list **students** into my "Enrollments.json" file [Example 5]. This was much more efficient than using the f-string method to write in each line of my **students** list into the JSON file [Example 6].

Example 5:

```
file = open(FILE_NAME, "w")
json.dump(students, file)
file.close()
```

Example 6:

```
file_obj = open(FILE_NAME, 'w')
for student in students:
   csv_data = f'{student[0]},{student[1]},{student[2]}\n'
   file_obj.write(csv_data)
file_obj.close()
```

Error Handling

Often, others break your code in unexpected ways. To make the process of recovering from Errors easier, error handling provides contextual information to the user about what went wrong when running the code. The program's developed can even customize the error message! This can help the user then troubleshoot on their end to achieve their desired result.

Exception Class

In Python, the Exception class will automatically create an Exception object which stores information about the error that occurred while the user was running the program. This can be really useful to convey to the user the technical information of what exactly went wrong. It is also useful so that both a simple error message and a more complex error message can be displayed to the user, allowing users of varying skill levels to understand exactly what went wrong. Users who may not have advanced skills in programming may also be to reach out to the program's developer with the technical error message for further help.

For my Assignment, I used the try-except construct and exception class to catch and handle errors [Example 7]. In this way, I was able to display both a generic error message and technical error message to the end-user [Figure 2].

Example 7:

```
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
# Error Handling
except Exception as e:
    print("ERROR: There was an error with reading your Enrollments file.")
    print("Please check that your Enrollments.json file exists.")
    print("-- Technical Error Message -- ")
    print(e.__doc__)
    print(e.__str__())
```

```
ERROR: There was an error with reading your Enrollments file.
Please check that your Enrollments.json file exists.
-- Technical Error Message --
File not found.
[Errno 2] No such file or directory: 'Enrollments.json'
```

Figure 2: Displaying Generic and Technical Error Message to the end-user

Summary

I learned the differences between lists and dictionaries, as well as the differences between JSON files and CSV files. For both dictionaries and JSON files, I learned how to create and append them using Python. For JSON files, I also learned how to read, import, and store the data into a variable in my code

using Python's json module. In addition to learning about dictionaries and JSON files, I also learned how to Error Handle via the try-except construct and Exception class.