Zach Harvey
11/13/2023
IT FDN 110 A
Assignment 5

Assignment 5 - Dictionaries and Exception Handling

Introduction

In this module, we began to use dictionaries, utilized the JSON library when working with files, and added exception handling to make our code more robust. Dictionaries are a good data structure for data access and editing. Paired with the Python JSON library, it was very simple to read and write to local files.

Dictionaries

In this assignment, we updated our “student_data” variable to be a dictionary. This allowed us to store newly entered data in key-value pairs as pictured in the image below.

```
student_first_name = input("Please enter the student's first name: ")
if not student_first_name.isalpha():
    raise ValueError("The first name should not contain numbers.")
student_last_name = input("Please enter the student's last name: ")
if not student_first_name.isalpha():
    raise ValueError("The last name should not contain numbers.")
course_name = input("Please enter the course name: ")

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

Image 1: Gathering user input and storing the data in a dictionary

A dictionary allows for a more explicit organization of data and more efficient data access when adding, removing, and reading from the dictionary. Below is an example of accessing values using the corresponding key.

```
for student in students:
    print(f'{student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}')
```

Image 2: Accessing dictionary values with corresponding keys

Because dictionaries and lists of dictionaries are formatted the same as JSON structure, we were able to leverage Python’s built-in JSON library to assist with reading and writing to local files. After importing the library, we used “json.load” and “json.dump” to read and write to a file, respectively. This cuts down on manipulating our previous data structure of a nested list quite a bit.

Exception handling

To be able to work effectively in Python and any other language, we must be able to capture exceptions or errors when they happen. In Python, we can use the “raise” syntax to instantiate an error with a certain type and message that can be caught. Built in methods will do the same without the programmer having to explicitly raise an exception. To catch these exceptions and alert the user, we can wrap our code in “try/except” blocks. This allows the code to be ran and continue if it succeeds without issue or catch and raise explicit exceptions as they arise. This pattern can be seen below. One thing to note is that we can catch certain types of exceptions or generically handle any exception that comes in.

```
try:
    student_first_name = input("Please enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError("The first name should not contain numbers.")
    student_last_name = input("Please enter the student's last name: ")
    if not student_first_name.isalpha():
        raise ValueError("The last name should not contain numbers.")
    course_name = input("Please enter the course name: ")

    student_data = {"FirstName": student_first_name, "LastName": student_last_name, "CourseName": course_name}
    students.append(student_data)
    continue
except ValueError as e:
    print(e)
    print("-- Technical Error Message --")
    print(e.__doc__)
    print(e.__str__())
except Exception as e:
    print("A non-specific error occurred")
    print("-- Technical Error Message --")
    print(e, e.__doc__, type(e), sep='\n')
```

Image 3: Exception handling

Summary

This module highlighted the benefits of deciding on which data structures to use and how they can work with different libraries. Additionally, understanding how to utilize exception handling is essential to developing robust programs.