Name: Brent Larson

Date: 02/11/2024

Course: Python 100

Assignment: Assignment05

GitHub:

Effective Data Management and Error Handling in Python

Introduction

This document explores the foundational concepts and advanced techniques that empower Python programmers to handle data. Beginning with a comparison between Dictionaries and Lists, we can have structured ways these data structures organize information, each providing advantages in specific scenarios. The practical application of JSON provides a seamless data interchange to encode and decode data effortlessly. As we explore structured error handling, the significance of the Try-Except pattern is explained, showing its scripts that manage exceptions. Finally learning of code files and management, the connections between traditional and modern practices ensures collaboration, code quality, and project easiness.

Dictionary vs List

The difference between a Dictionaire and List is how they organize and access data. Lists are collections where each item is indexed numerically. Each list starts with zero and can be accessed using the number the item is positioned in the list. Dictioinaires, use key-value stem for quick data retrieval by key rather than by index position. Dictionaires are useful for storing structed data, similar to rows in a database or spreadsheet, where each row can represent an entity with attributes defined by keys. The choice between using a List or Dictionary often depends on the need of your program, such as whether order of items or quick access by name is more valuable.

I have highlighted two ways to access an item in a List and Dictionary below. The significant difference is I will access the index in the list, in this example ice_cream[1] is “Chocolate”. In the dictionary I will access the key to find the name of the ice cream, ice_cream[“name”] which is Chocolate.

List

```
# Creating a list
ice_cream = ["Vanilla", "Chocolate", "Mint", "Strawberry"]
# Accessing the second item
print(ice_cream[1])
```

Dictionary

```
# Creating a dictionary
ice_crem_dict = {"name": "Chocolate", "color": "brown", "taste": "coco"}
# Accessing the value with key "name"
print(ice_crem_dict["name"])
```

Writing data to a Dictionary

Writing from a file into a dictionary involves reading each line of the file, processing the data to split it into parts that correspond to the keys and values and then creating a dictionary with this information. Data can be read from a file, with each row representing a “row” of data similar to a database or spreadsheet row. This row is then split based on a delimiter, for example a comma or space. This delimiter will separate each piece of data, which will be used to create a dictionary. This dictionary is then appended to a list, creating a list of dictionaries, each representing a row from the file. In summary this method involves opening the file in read mode, iterating over each line, using the split method to divide the line into parts, creating a dictionary for each line, and then appending this dictionary to a list.

JSON (JavaScript Object Notation)

A JavaScript Object Notation file is a lightweight data file format that is easy for programmers to read and write, and easy for applications to parse and generate. JSON is based on the JavaScript programing language, but its usage has expanded beyond JavaScript to become a generic format used in many programming languages for data exchange. JSON Files are text-based, with syntax consisting of objects and arrays making it very readable. The format is designed to represent simple data structures and arrays called objects, which have text being the common thread. JSON is useful for web applications and APIs, where it can facilitate the exchange of data between servers and clients. It supports a range of data types, including strings, numbers, Booleans, arrays, and objects. JSON files typically have the extension .json and consist of key-value pairs.

```
import json
# Open a JSON file for reading
file = open("data.json", "r")
data = json.load(file)
# The JSON "data" now shows as a Python list of dictionaires
for item in data:
    print(f"key: {item["key"]}, value: {item["value"]}")
```

JSON Modules

Python’s JSON modules provide a straightforward way to encode and decode JSON data. It comes with a set of functions or syntax for parsing JSON strings. The primary functions are:

“json.load(f)”: This reads and decodes JSON data from a file like object “f” which converts it to Python object. The function is commonly used for reading JSON files.

“json.loads(s)”: Parces a JSON string “s” converting it to a Python object. This is useful for receiving JSON strings or data from an API.

“json.dump(obj)”: Encodes a python object into a JSON formatted string. This method is good for generating JSON to be sent to a file or API.

Structured Handling

Structured handling is often referred to as the Try-Except. This name comes from exactly the syntax used in your code, try and except. It is similar to an if-else statement as it follows the code and allows you to add an error message depending on the written code. Error handling improves any written scripts by managing errors your program may encounter. It allows the programmer to write messages that help the user navigate what may be asked or the program to run more smoothly.

Try block- The code that might raises an exception is placed inside this block of code. If an exception is specified using try, it runs what is inside the try block.

Except block – This block is used to catch and handle the exception(s) that are specified in the try block. Multiple except blocks can be executed.

Else block – This may be used if a programmer wants to specify code that should run if the try block does not raise an exception.

Finally block – This block runs no matter what, whether the exception is executed or not. It is typically used for closing files or networks.

One error I ran into while trying to write script this week was reading an empty file. Here is the **except** block I used.

```
except (FileNotFoundError, json.decoder.JSONDecodeError):
    students = []
    print(f"File {FILE_NAME} not found, starting with empty list")
```

Code Files and Management

Saving and sharing code files are essential in software development for enhancing collaboration, efficiency, and code quality. These practices allow multiple developers to simultaneously contribute to a project, promote code reusability across projects, and facilitate knowledge sharing within teams. Early error detection through peer reviews leads to higher code quality, while version control systems like Git track changes, offering insights into the evolution of the codebase. Storing code in centralized locations, such as cloud services or version control platforms like GitHub, ensures backup and facilitates disaster recovery. Additionally, these methods support remote work, comply with industry regulations, and enable contributions to open-source projects. Modern code management practices, encompassing both traditional network shares and cloud-based solutions, are pivotal for maintaining project momentum and fostering an environment of continuous learning and improvement.

Network File Sharing

A traditional method allowing team collaboration through a central storage space, akin to a shared folder accessible within an organization's network.

This type of file sharing is commonly used in professional settings such as a business. Since networks can store large amounts of data.

Cloud File Sharing
This is a modern approach utilizing cloud services for remote access and collaboration on files. Many different type of cloud storage providers exist, including Google, Dropbox, OneDrive, and Amazon S3. This option provides scalability, data redundancy, along with accessibility.

GitHub

GitHub is a cloud-based platform that is tailored for code hosting and collaboration, that includes file sharing within the software development context.

Features of GitHub
File Hosting: Supports various file types alongside code.
Version Control: Utilizes Git for change tracking.
Collaboration: Offers tools like pull requests and issue tracking.
Access Control: Manages user permissions for repository access.
Cloud-Based Access: Enables web-based repository management.

Using Dictionaires, Files, and Exception Handling in my Python Script

Dictionaries for Storing Student Information
I used dictionaries to store each student's information, including their first name, last name, and course name. This choice allows for a structured and intuitive way to handle individual student records.

```
while not course_name:
    course_name = input("Please enter a valid course name")
new_student = {"first_name": student_first_name, "last_name":
student_last_name, "course_name": course_name}
students.append(new_student)
```

File Handling Using JSON
For saving and loading student data, I employed file handling techniques, specifically with JSON files. JSON is a lightweight data interchange format that's easy to read and write for humans and machines.

```
# Saving to a file
with open(FILE_NAME, "w") as file:
    json.dump(student_data, file)
# Loading from a file
with open(FILE_NAME, "r") as file:
    student_data = json.load(file)
```

Exception Handling
When attempting to open and read from "Enrollments.json", the code uses a **try** block to catch these specific exceptions. If the file doesn't exist (**FileNotFoundError**) or contains invalid JSON (**json.decoder.JSONDecodeError**), the program prints a message indicating the file was not found and starts with an empty list of students, preventing the program from crashing and allowing it to continue running smoothly.

```python
try:
    with open("Enrollments.json", "r") as file:
        student_data = json.load(file)  # Attempt to load JSON data
except FileNotFoundError:
    print("File Enrollments.json not found, starting with an empty list.")
    students = []  # Initializes an empty list if file is not found
except json.decoder.JSONDecodeError:
    print("Error decoding JSON from file. Starting with an empty list.")
    students = []  # Initializes an empty list if JSON data is invalid
```

Conclusion

Reflecting on the comprehensive exploration of data management and error handling in Python, it's evident that these aspects are an asset in software development. The understanding of when to employ Dictionaries over Lists, the utilization of JSON for data interchange, and the application of Python's JSON modules are pillars to Python code. The adoption of structured error handling such as the Try-Except pattern exemplifies Pythons approach to software readability, minimizing disruptions and enhancing user experience. As we look at file-sharing methods of cloud-based solutions, the importance of adaptable code management practices is essential. This guide through the basics of Python programming not only elevates our technical understanding of data management and error handling and their key to innovation and success.