11/15/2024

Maheep Chawla

100909435

Scripting

Assignment 2

Table of Contents

[**Problem 1: Sorting and Arranging Files in a Directory** 2](#_Toc182528953)

[**Description** 2](#_Toc182528954)

[**Value and Usefulness** 2](#_Toc182528955)

[**Difficulty Level and Expected Libraries** 2](#_Toc182528956)

[**Solution** 3](#_Toc182528957)

[**Reflection** 3](#_Toc182528958)

[**Problem 2** 5](#_Toc182528959)

[**Description** 5](#_Toc182528960)

[**Value and Usefulness** 5](#_Toc182528961)

[**Difficulty Level and Expected Libraries** 6](#_Toc182528962)

# **Problem 1: Sorting and Arranging Files in a Directory**

**Description**

Among the most common problems which one comes across is a messy directory containing any number of files that may be different in terms of file type, size, or modification date. Sorting them out manually requires immense time, especially since, in professional and academic life, the number of files in the directories can reach hundreds and sometimes even thousands of files. This Python script is designed to sort these files into subfolders according to various sorting methods such as filename, size of the file, modification date, or file extension.

Examples might include sorting based on "filename," placing files alphabetically and into alphabetically-named folders, such as A-Files, B-Files, or the like. Another option could be "filesize," sorting to folders such as "Small," "Medium," and "Large." Sorting by "date modified" could place them into folders by year or month.

This kind of automation is useful in that it saves users from having to dig around for files and prevents digital clutter from building up, making the directory much more navigable and manageable. The categorization of files based on type, date, or size has beneficial implications in practical fields that demand tight data management and accessibility of files, such as digital marketing, research, and data analysis.

## **Value and Usefulness**

A solution like this would greatly assist casual and professional users working with a lot of files on a regular basis. An automated file organization tool will allow users to automate that part of the workflow, freeing time for more productive manual tasks. What's more, it is also a good demonstration of Python's file handling and automation capabilities, hence an always impressive addition to a developer's repertoire.

## **Difficulty Level and Expected Libraries**

This problem is moderate, considering that it requires a good amount of Python file handling, sorting, and some light error handling. Major libraries used will include:

* **os**: The standard way to go for navigation in the file system, checking file types, and retrieving file sizes.
* **shutil**: Moving files into categorized folders.
* **datetime**: To perform date sorting and to convert timestamps to readable formats.

# **Solution**



# **Reflection**

This file sorting/organizing application was an interesting challenge. The task sounded quite simple at first-sort files according to certain criteria like name, size, date modified, or extension. However, starting the development of the solution showed me that in order for files to really be moved and organized into folders according to this or that criterion, more detailed logic and robust handling of files and directories were needed.

**Did you pick a suitably challenging problem to solve? Or was it too easy, or too hard?**

I do feel the problem was challenging enough to be in this position yet not too complex to be out of place in helping me further my skills in Python file handling and organization. Although not very complicated, it did require some forethought to make certain each of the sorting criteria functioned correctly and did not mess with the overall directory structure. Adding the capability for creating folders for files depending on the sorting criteria one decides did raise the complexity, making the project a well-rounded one.

**What was the most challenging aspect of solving this problem?**

The hardest part of the whole problem was to perform an actual movement of physical files correctly based on different criteria. For example, carrying out the sorting of the files by size and grouping them into folders like "Small," "Medium," and "Large" included introducing meaningful thresholds on the size and processing peculiar cases of files which either have roughly the same or the same size. Another challenge was sorting the files by their modification date, including extracting the dates and formatting them into a meaningful structure to create folders for sorting, all while showing that the files would fit nicely into these date-named folders. Further, handling cases of files having no extension, or the directory contained files with identical names, also requires some mindful logic to avoid errors or overwriting of data.

**What resources did you use to learn about the libraries (or any other new features) you used?**

The realization of this project has been based on the os and shutil libraries that I knew to some extent; however, some deepening was necessary with respect to certain functions. I consulted the official Python documentation of os.path and shutil regarding the existence of such specific methods as os.path.getmtime() for sorting by date and shutil.move() for safely moving files across directories. Resources such as the Python documentation and online forums helped shed light on this question around edge case handling related to file manipulation, and I also reviewed best practices in creating and managing dynamic directories based on set criteria.

**What was the most valuable thing you learned from this assignment?**

One of the best takeaways from this assignment was error handling in dealing with file systems. I learned to expect potential errors-most notably, nonexistent directories, empty folders, and files without extensions-and gracefully handle them so the application would not break. Also, separating the code into functions to sort files and move them made the solution more modular and reusable; this enabled me to test pieces of code independently, further reinforcing good programming practices.

**How did you test your completed product? How confident are you that it works reliably?**

Testing consisted of the creation of a sample directory with files of different names, sizes, dates of modification, and extensions. I wanted to test each option of sorting and make sure it works as expected. I run each criterion several times to confirm that the files were sorted into appropriate folders without data loss or misplacement. Since the solution was consistent under all my test cases, I am fairly confident that it is reliable. However, I would suggest more tests on larger directories to ensure that the code scales and performs well.

**Is there something you would still like to add to this, or something it makes you want to try next?**

Even though the program performed very well for its purpose, I did have a couple of suggestions for enhancements. First and foremost, a GUI would make this utility way easier for people who are not comfortable directly working with the command line. It would also be interesting to extend the program further so that the sorting can be expanded upon by the user himself, which could also be extended by selecting size limits or other parameters based on metadata, such as date created or file author if the file is a document. These enhancements will make this tool all the more flexible and useful for any purpose one may have in mind.

**Did you *enjoy* the experience of working on this assignment? Why?**

All in all, this project was enjoyable, as it had a practical application that might prove to be really useful for any personal file management needs. It was nice to see the program arrange the files in a nice fashion; thus, an unorganized directory became a nicely organized one. This further fueled my view of how automation simplifies tasks and made me confident with using Python for useful file-handling tasks that might be useful in personal and professional scenarios.

# **Problem 2**

## **Description**

JSON or JavaScript Object Notation is a light format of data that is human-readable. It finds a wide application in storing, transmitting, and processing of data. The JSON format is excellent for API responses and data exchange but is not really an end-user-friendly format as far as analysis or reporting is concerned for users more comfortable working with spreadsheet applications. In many cases, APIs, web scraping, and databases return information in JSON format, but for analysis or reporting purposes, these data sets are more usefully deployed in tabular formats such as Excel or CSV.

This conversion task is shared by Data Analysis, Business Intelligence, and Software Development in terms of converting JSON files into Excel or CSV format. Below is a Python script that would automate the process of converting a predefined JSON file into Excel or CSV format for easy access and manipulation in tools like Microsoft Excel or Google Sheets. The program should execute the extraction of data from JSON, parsing, and then converting into neat tabular structure that can be analyzed further.

This process would entail the following:

**Reading the JSON File**: This will employ Python's in-built JSON library to read and parse the file. Flattening Nested JSON: JSON objects can be nested, meaning that your data may have many levels-in other words, nested dictionaries. These must be flattened to make them compatible with a tabular format.

**Converting to CSV/Excel**: Designing the data, using libraries like pandas, into rows and columns, and exporting it as CSV or Excel format.

**Handling Edge Cases**: The code must be in a position to handle the different structures that appear within JSON data; these may be different in cases of no values or nested arrays, data types that require formatting.

For example, a JSON file with a list of users with details such as name, age, and address that includes street, city, and zipcode will be parsed to a CSV/Excel format where every user is a row and every attribute is a column: name, age, street, city, and zipcode.

## **Value and Usefulness**

Conversion of JSON into Excel or CSV format has plenty of real-life applications in data science, business intelligence, and reporting. Here are a few major benefits:

* Data Analysis: Data scientists and analysts working with any dataset in JSON format often feel comfortable working with tabular data formats like Excel or CSV for analytics, visualization, and reporting.
* Interoperability: Many tools, databases, and reporting software support Excel and CSV format. Converting JSON into these formats ensures that the data will be shared and consumed with ease.
* Productivity: In fact, JSON file conversion automates conversions for the user and saves them lots of time, especially when datasets are huge. For example, a user may receive large datasets in JSON but would want or need it in an accessible form, tabular fashion, for everyday use.

A solution like this is not only solving one very common data formatting problem for the developer/business professional, but it also shows proficiency in Python and the manipulation tools so integral to both personal and professional projects.

## **Difficulty Level and Expected Libraries**

Since this task is of a moderate complexity, it includes file I/O operations, manipulating different data formats such as JSON, CSV, and Excel, and providing support for any edge case that could come up because of nested structures or missing data. Now, here's a breakdown of the utilized tools and libraries:

* **json Library**: Python's standard library is to be utilized for loading and parsing. It makes reading in the JSON format quite easy.
* **Pandas**: This is a very strong library for data handling or manipulation. It will be used to convert the data into a DataFrame from where a .xlsx or a .csv could easily be written. Pandas makes it really convenient to handle and flatten nested JSON structures into a tabular format.
* **openpyxl or xlsxwriter Library**: These libraries basically deal with reading and writing Excel files. They will be needed if the user wants to export the data into Excel format.
* **csv Library**: Python's standard library for reading and writing CSV files. It will be used in view places where the format of output is in CSV.