Adam Packer

March 18, 2025 (Extended due date via student hardship)

UW: IT FDN 110 A Wi 25: Foundations Of Programming: Python

Module 5 Assignment

Assignment 5: Working with Data Structures, File Handling, and Error Handling in Python

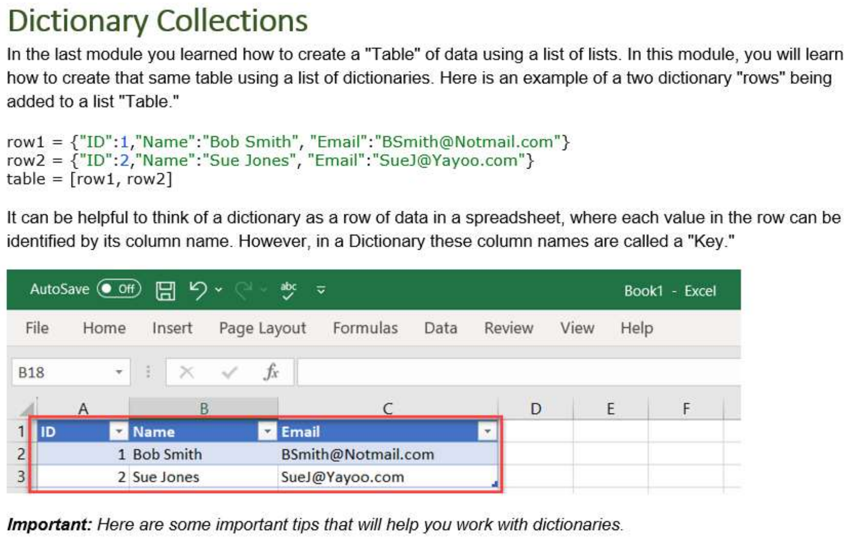
Introduction

Assignment 5 reviewed several key topics that I am used to working with in the context of my employment, for example File Handling and Error Handling. However, since my work primarily exists in Excel and VBA, learning how to properly manage file input / output with Python, as well as structure my error handling correctly in Python, are key things I need to learn. I am looking forward to learning more about Dictionaries as well.

List of Lists versus a List of Dictionaries

In the past module I was very excited about the concept of a list of lists because it was similar to how I am used to thinking of data in a grid format in Excel, and a very large part of this course for me is thinking about how I can leverage my former work in Excel to build Python-based tools. Now I am seeing that this module builds on lists of lists in the form of lists of dictionaries. This is even more exciting to me because the concept of a dictionary seems to allow for really targeted identification of a data point in a set. This is important because a lot of the work I do in Excel is fundamentally identifying data points in some kind of structured data set. Since lists have a specific order and can be added to, lists seem like a good starting point, but really the value will be in lists of dictionaries.

Figure 1: The course notes draw a specific reference to Excel



Dictionary Collections

In the last module you learned how to create a "Table" of data using a list of lists. In this module, you will learn how to create that same table using a list of dictionaries. Here is an example of a two dictionary "rows" being added to a list "Table."

```
row1 = {"ID":1,"Name":"Bob Smith", "Email":"BSmith@Notmail.com"}
row2 = {"ID":2,"Name":"Sue Jones", "Email":"SueJ@Yayoo.com"}
table = [row1, row2]
```

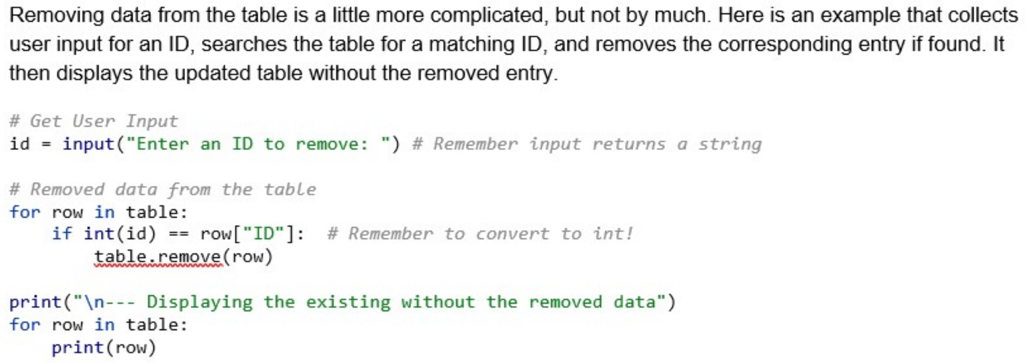
It can be helpful to think of a dictionary as a row of data in a spreadsheet, where each value in the row can be identified by its column name. However, in a Dictionary these column names are called a "Key."

	A	B	C	D	E	F
1	ID	Name	Email			
2	1	Bob Smith	BSmith@Notmail.com			
3	2	Sue Jones	SueJ@Yayoo.com			

Important: *Here are some important tips that will help you work with dictionaries.*

The key is the ability to add and remove data that is uniquely identified in the data set. I have worked with some tools that I think were based on SQL that made it difficult to delete a record once it had already been established. One tool in particular was very annoying because any time you made a record it was present for all of time no matter what. You could mark it as deprecated or irrelevant, but you couldn't delete the record even if it was a total meaningless mistake. I don't want to build systems that do that.

Figure: Course Notes show how to remove the table



Removing data from the table is a little more complicated, but not by much. Here is an example that collects user input for an ID, searches the table for a matching ID, and removes the corresponding entry if found. It then displays the updated table without the removed entry.

```
# Get User Input
id = input("Enter an ID to remove: ") # Remember input returns a string

# Removed data from the table
for row in table:
    if int(id) == row["ID"]:  # Remember to convert to int!
        table.remove(row)

print("\n--- Displaying the existing without the removed data")
for row in table:
    print(row)
```

Using Dictionaries with File Data

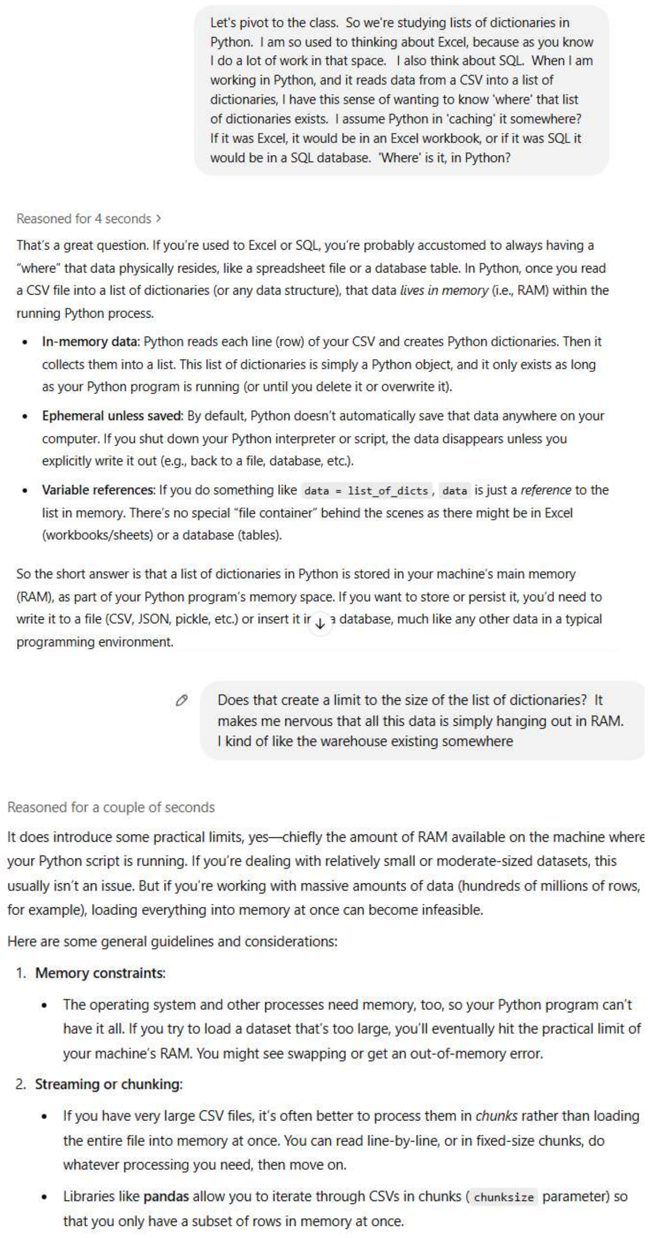
In my industry, there is a constant exchange of a variety of data files so it is important for me to see in the context of Python how to do this. As we've discussed, for me a lot of this will be reading CSV files into a dictionary, so that is what I am focused on versus reading a TXT file into a dictionary.

One portion of the Module 5 notes that I really like was the comparison of lists and dictionaries in the context of data organization, accessing data, data structure choice, data validation, data transformation, and field names. After reading the comparison, for my work it seems to be the clear choice to use dictionaries because of how the data is specifically associated with a key value pair. It will be interesting to see whether there are examples of where it would still be better to use a list of lists, versus dictionaries. I need to be able to reference specific data points really clearly in my calculations, and it seems that dictionaries will be the best option. That said, I am also interested to learn, at some point, the same kind of contrast between using lists of dictionaries versus structuring data in SQL, and then using SQL queries to pull the data. There is this long standing debate whether at some point we've pushed Excel outside the parameters of what it was best intended for, in a way that starts to look like we want it to be SQL. I think the counter argument is strong – that Excel is perfectly capable of what we want to do and SQL is actually more opaque. To extent that conversation, I like how Python structures this process of data organization and I don't necessarily see the reason to jump to SQL. But I wonder if SQL afficionados would suggest that we're using Python in ways that were better suited to SQL. I am on team Python for sure, but I like knowing the debate.

Discussing Data Structure Parameters with ChatGPT.

As we're discussing reading CSV files into memory with Python, I am continually having the thought (bias) that I expect there to be some kind of place, some data warehouse, where the data resides and then Python can operate on it. It feels very intangible to me that Python would be reading a CSV file into a list of dictionaries, and then what? – just holding it memory? I turned ChatGPT to help me wrap my head around this. I am used to Excel, or even SQL, where the data sits in a particular place and logic operates on it. At least that is how I have it in my head. It was an interesting conversation:

Figure 3: Conversation with ChatGPT on how I should think of where the data in Python 'lives'



Let's pivot to the class. So we're studying lists of dictionaries in Python. I am so used to thinking about Excel, because as you know I do a lot of work in that space. I also think about SQL. When I am working in Python, and it reads data from a CSV into a list of dictionaries, I have this sense of wanting to know 'where' that list of dictionaries exists. I assume Python in 'caching' it somewhere? If it was Excel, it would be in an Excel workbook, or if it was SQL it would be in a SQL database. 'Where' is it, in Python?

Reasoned for 4 seconds >

That's a great question. If you're used to Excel or SQL, you're probably accustomed to always having a "where" that data physically resides, like a spreadsheet file or a database table. In Python, once you read a CSV file into a list of dictionaries (or any data structure), that data *lives in memory* (i.e., RAM) within the running Python process.

- **In-memory data**: Python reads each line (row) of your CSV and creates Python dictionaries. Then it collects them into a list. This list of dictionaries is simply a Python object, and it only exists as long as your Python program is running (or until you delete it or overwrite it).
- **Ephemeral unless saved**: By default, Python doesn't automatically save that data anywhere on your computer. If you shut down your Python interpreter or script, the data disappears unless you explicitly write it out (e.g., back to a file, database, etc.).
- **Variable references**: If you do something like `data = list_of_dicts`, `data` is just a *reference* to the list in memory. There's no special "file container" behind the scenes as there might be in Excel (workbooks/sheets) or a database (tables).

So the short answer is that a list of dictionaries in Python is stored in your machine's main memory (RAM), as part of your Python program's memory space. If you want to store or persist it, you'd need to write it to a file (CSV, JSON, pickle, etc.) or insert it ir ↓ a database, much like any other data in a typical programming environment.

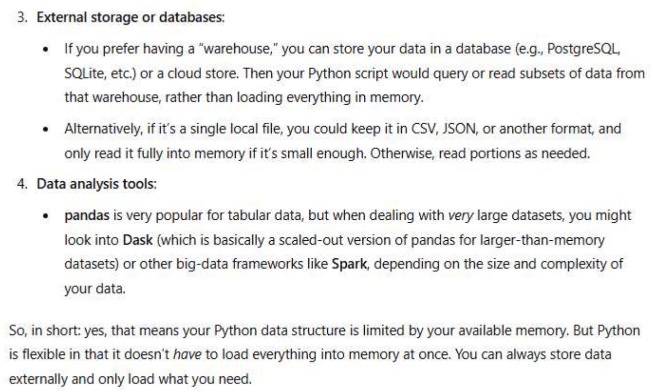
Does that create a limit to the size of the list of dictionaries? It makes me nervous that all this data is simply hanging out in RAM. I kind of like the warehouse existing somewhere

Reasoned for a couple of seconds

It does introduce some practical limits, yes—chiefly the amount of RAM available on the machine where your Python script is running. If you're dealing with relatively small or moderate-sized datasets, this usually isn't an issue. But if you're working with massive amounts of data (hundreds of millions of rows, for example), loading everything into memory at once can become infeasible.

Here are some general guidelines and considerations:

1. **Memory constraints**:
    - The operating system and other processes need memory, too, so your Python program can't have it all. If you try to load a dataset that's too large, you'll eventually hit the practical limit of your machine's RAM. You might see swapping or get an out-of-memory error.
2. **Streaming or chunking**:
    - If you have very large CSV files, it's often better to process them in *chunks* rather than loading the entire file into memory at once. You can read line-by-line, or in fixed-size chunks, do whatever processing you need, then move on.
    - Libraries like **pandas** allow you to iterate through CSVs in chunks (`chunksize` parameter) so that you only have a subset of rows in memory at once.



3. **External storage or databases:**
   - If you prefer having a "warehouse," you can store your data in a database (e.g., PostgreSQL, SQLite, etc.) or a cloud store. Then your Python script would query or read subsets of data from that warehouse, rather than loading everything in memory.
   - Alternatively, if it's a single local file, you could keep it in CSV, JSON, or another format, and only read it fully into memory if it's small enough. Otherwise, read portions as needed.
4. **Data analysis tools:**
   - **pandas** is very popular for tabular data, but when dealing with *very* large datasets, you might look into **Dask** (which is basically a scaled-out version of pandas for larger-than-memory datasets) or other big-data frameworks like **Spark**, depending on the size and complexity of your data.

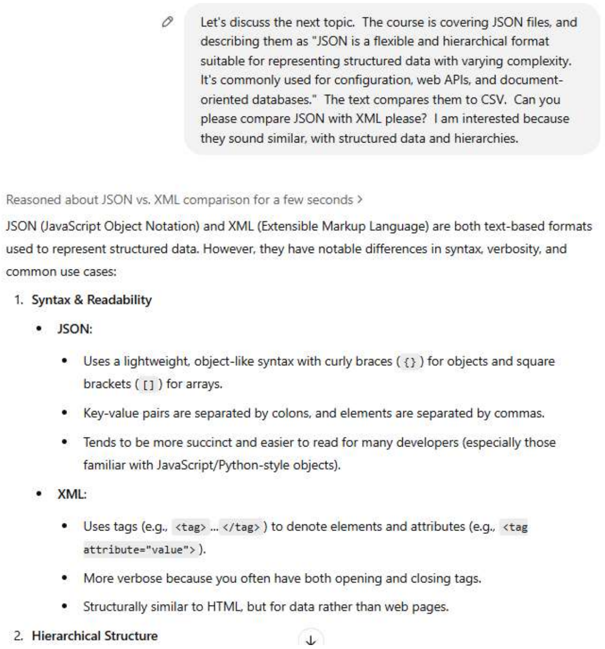
So, in short: yes, that means your Python data structure is limited by your available memory. But Python is flexible in that it doesn't *have* to load everything into memory at once. You can always store data externally and only load what you need.

I thought that ChatGPT's answer made sense in terms of how I should think about it. I can't get out of my head the preference that the data should live somewhere less ephemeral than being read into RAM. Sure, at the end I can draft logic to re-write the material to a CSV file, and it will exist there. I would be interested to know how 'odd' it is that I prefer this. Is everyone okay with their dataset existing in this transitory state? Not to be too clever, but AITAH?

JSON vs. CSV vs. Adam's focus XML

The course text has a section that compares JSON to CSV, and takes some time to explore the differences, including the fact that JSON can have hierarchical data structured into the JSON file, whereas the CSV file is obviously a flat file. That immediately made me think of XML format, which is very common in the tax industry. Governments all over the world rely on XML format for the format of the tax return data that taxpayers use to file tax returns. The data is specifically tagged, and the XML schema is carefully formatted by the government authority, so that whatever computer system produces it must produce it in the valid format. I was interested to compare and contrast JSON files with XML files, and I asked ChatGPT to develop this for me in more detail:

Figure 3: Conversation with ChatGPT on JSON versus XML files (portion)



Let's discuss the next topic. The course is covering JSON files, and describing them as "JSON is a flexible and hierarchical format suitable for representing structured data with varying complexity. It's commonly used for configuration, web APIs, and document-oriented databases." The text compares them to CSV. Can you please compare JSON with XML please? I am interested because they sound similar, with structured data and hierarchies.

Reasoned about JSON vs. XML comparison for a few seconds >

JSON (JavaScript Object Notation) and XML (Extensible Markup Language) are both text-based formats used to represent structured data. However, they have notable differences in syntax, verbosity, and common use cases:

1. **Syntax & Readability**
   - **JSON:**
     - Uses a lightweight, object-like syntax with curly braces (`{}`) for objects and square brackets (`[]`) for arrays.
     - Key-value pairs are separated by colons, and elements are separated by commas.
     - Tends to be more succinct and easier to read for many developers (especially those familiar with JavaScript/Python-style objects).
   - **XML:**
     - Uses tags (e.g., `<tag>` ... `</tag>`) to denote elements and attributes (e.g., `<tag attribute="value">`).
     - More verbose because you often have both opening and closing tags.
     - Structurally similar to HTML, but for data rather than web pages.
2. **Hierarchical Structure**

Like the course text, ChatGPT seems to focus on JSON as lightweight, easy to read / use, and that developers like it. I am interested whether that will work for me. I am REALLY used to thinking about tax data in the structure of the XML format, and the existence of XML data tags. XML data tags are really useful, and because the government produces the XML schema for tax returns, everything sort of revolves around that XML. It HELPS that the government produces it, but at the same time you don't have a choice, at least with regard to tax returns, to have the ENDING structure of the data really be any different.

All that said, I liked the course comparison of CSV and JSON. I have helped design systems that use CSV file as an upload. I was the developer on it, but that's how we built it. So I am used to thinking of CSV files that way. I think having JSON as an additional tool in the tool kit is going to be important if I want to have a lighter format than XML, and something with more hierarchical data organization than CSV. That said, I'll need to think a little more about where this 'happy medium' could really fit into my system development. I'll probably end of defaulting to either CSV or XML, but I will try to use JSON so that I have a well-rounded approach.

Structured Error Handling: Try – Except

Error handling is a critical part of the work that I already do, so this section of the Module made a lot of sense. This includes not just basic error handling that the User might not see, for example wrapping an Excel formula in =iferror(formula, 0) so that if the result of the nested formula is an error the result of the of the outer formula is 0. That is obviously problematic because it doesn't really tell the User that a problem has happened. For that reason I try not to use it, but in some cases I still use it if I know that no matter what the kind of error is, I want the result to be zero if that happens.

I like that the try-except framework has multiple options for highlighting to the User not only that an error has occurred, but to tell the User more specifically what the error is.

Drafting the Code

I took a similar step-by-step approach to drafting the assignment code. Predictably, it required a lot of trial and error, including going back and rewriting sections of the code as it didn't work. I was also struggling a little with PyCharm, which started to feel kind of not User friendly to me.

After many rounds of trying, and working with ChatGPT to fix certain mistakes I was making, I got the code to work.

Summary

I found this Module to be a little harder than some of the past Modules. That said, some of the topics that were covered were key topics for me, including the use of Dictionaries. I think that a lot of the work that I am going to be doing in Python will involve dictionaries rather than lists, so this introduction was very important for me. Also, I really enjoyed learning about the way that Python has structured error handling. In my Excel work I layer in very detailed error handling, so this is something that I think will be an important part of my work moving forward.