WebAPI Call Assignment for Federal Spending

To test the knowledge obtained from the course in using a WebAPI call in python, we will use an api to get USA spending, https://api.usaspending.gov/. I am giving the specific Federal Account for this assignment, but feel free to explore more of the other sources available.

The base repo to pull down is https://github.com/awilbourn-eastern/WebApiFederalSpending.git, you will not commit changes back, so you may just want to fork the project.

Recall, that when calling WebAPI there are often paginated reports, meaning you cannot get all the data in one pass and must walk the results to get the next page of data. The accounts picked do have two pages to get all the data, but you are to pull the same data for the fiscal years of 2020 and 2021.

You are to build the application so that it will receive the parameters of the accounts and fiscal years from a command line argument to be passed. So, the values should be considered lists in each case to pass. Refer to the lecture demo on how a list can be passed. The command line arguments are to be named **accounts**, **fiscal_years**, and **deleteExistingDB**. Name your python file **getFederalSpendingData.py**, this will allow for automated testing and other years and accounts can be passed to ensure the algorithm works no matter what account or fiscal years are chosen. The **deleteExistingDB** should default to a value of **False**, but if containing a true value the code should handle deleting the database file, or just clear the tables of the database, and then continue to use the call in each run of the application to create the tables is they do not exist (schema code is provided).

Sample URLs

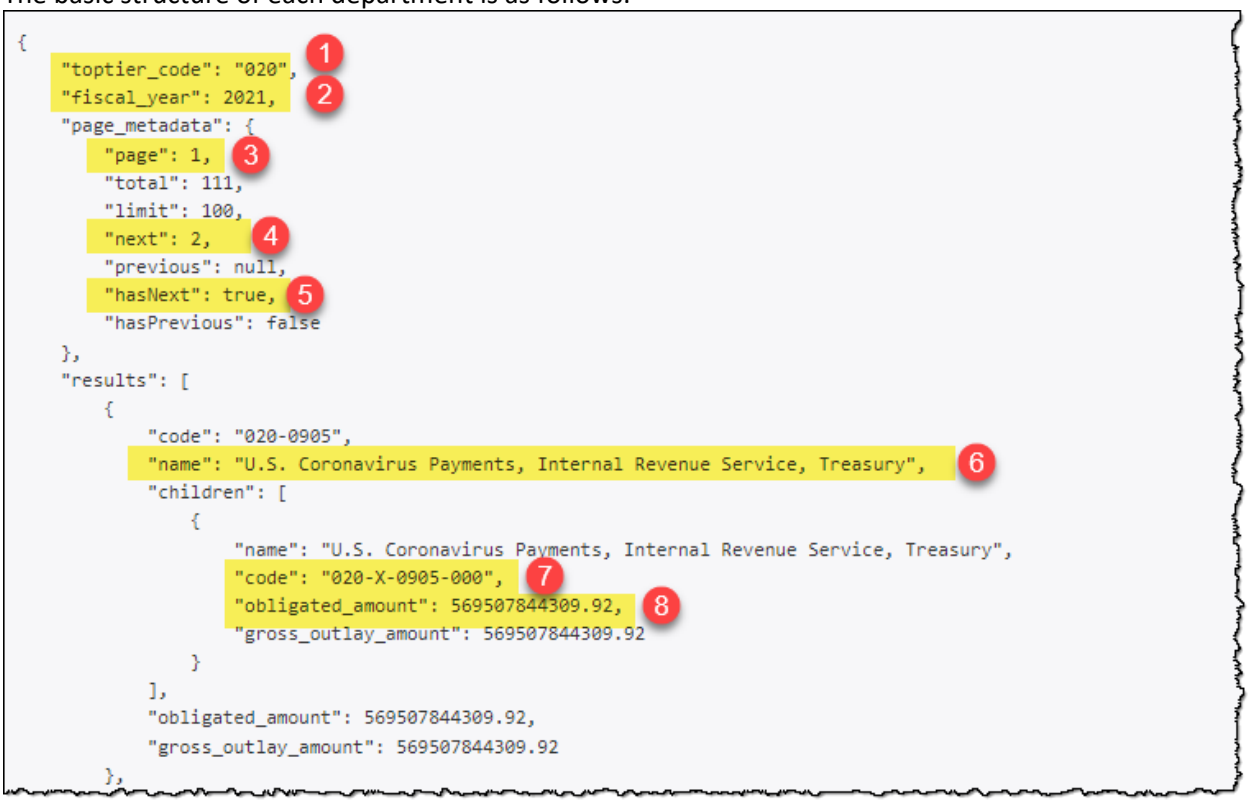
Highlighted below are the key aspects that will be filled in, the page numbers are to be determined back on the data returned as shown in the **page_metadata** section. As a hint, the "next" attribute will contain a **null** if there no other pages to walk, or to look at the "hasNext" will be **false**.

Account 012

https://api.usaspending.gov/api/v2/agency/012/federal_account/?fiscal_year=2021&limit=100&page=1
https://api.usaspending.gov/api/v2/agency/012/federal_account/?fiscal_year=2021&limit=100&page=2

Account 020
https://api.usaspending.gov/api/v2/agency/020/federal_account/?fiscal_year=2021&limit=100&page=1
https://api.usaspending.gov/api/v2/agency/020/federal_account/?fiscal_year=2021&limit=100&page=2

The basic structure of each department is as follows:



Significance of numbered labels in screen shot, presume data stored in object called **data**:
NOTE: the results example is just syntax to get the first item for example only

1. data[“toptier_code”]: Indication of the main federal agency account the data shows.
2. data[“fiscal_year”]: The year the data returned contains
3. data[“page_metadata”][“page”]: the current page being reviewed
4. data[“page_metadata”][“next”]: the next page to pull to get data
5. data[“page_metadata”][“hasNext”]: indication if there is another page of data to pull results for
6. data[“results”][0][“name”]: this is the name of the department
7. data[“results”][0][“children”][0][“code”]: the child department code
8. data[“results”][0][“children”][0][“obligated_amount”]: the amount for each child department used for answering the questions for amounts asked

The assignment is to import all the data for the **accounts** and **fiscal_years** that are passed into the application and store the data in a database. This will ensure values are not hard coded in the results to return. Provided with the given SQLite schema, store the data in an SQLite database and use queries to obtain the answers. In some cases, a dataframe is returned, in others it is just a single value only as indicated. Think about building this in a modular, reusable, chucks of code.

If a specific account in asked in the question verify the account data exists first, if not found in the data return a message like `"{account} is not found in the data"` replacing the appropriate value that may have been passed in. To be clear, if account 075 was passed in on the command line rather than 012 and 020, then questions related to the accounts 012 and 020 will not be able to be answered, or child department details. Therefore, checking the values of what was loaded in the data prior to querying for

the database is needed, as simple check if an account number exists and return true or false will work and if false the message as indicated.

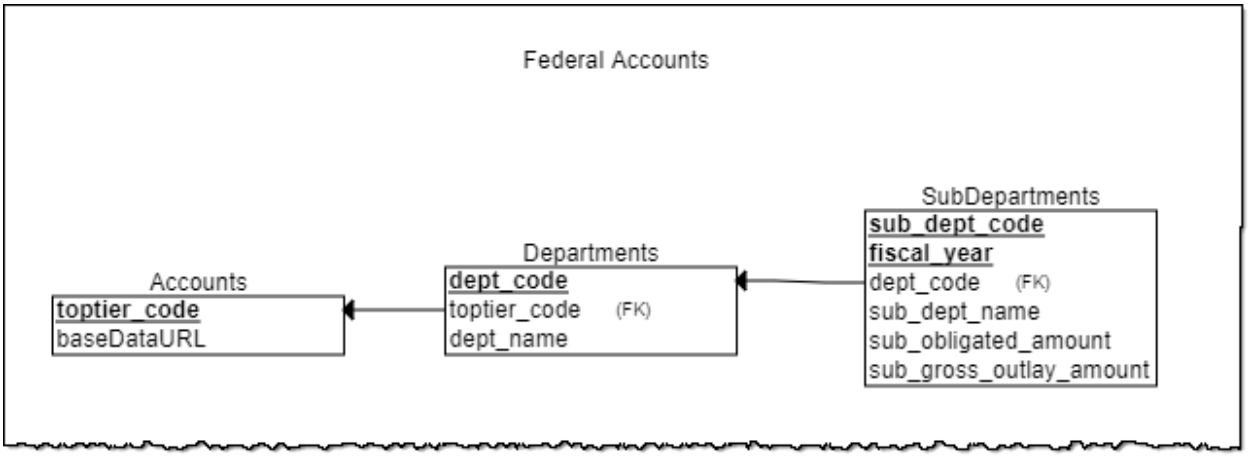
To obtain the answers add code to function shells in the form of getQuestion[x]Answer, the [x] being the question number. Example: `getQuestion1Answer() - getQuestion8Answer()`

The data is to be loaded by calling the main function and ensure to parse the arguments passed to build an algorithm to be able to load the data based on the parameters and results returned.

SQlite Schema

Like the demo lecture, create a way to build the SQlite database structure. Each table would be filled as the data is retrieved and then querying of the data is how answers are returned. The underline are the primary keys of each table, so duplicate values are not allowed.

NOTE: The table of accounts is not as beneficial as we have minimum other attributes to store, but the exercise is to help reinforce how relationships exist in databases. So at minimum an INSERT is required for each account passed in as an argument and the core URL for that given account. An example is https://api.usaspending.gov/api/v2/agency/020/federal_account/. Think of a base URL in a config settings.json to read from and replace the yellow for each account to be written to the database. A set of functions were created similar to the lecture as a starting point, minus the ability to delete the existing database if needed.




```
CREATE TABLE IF NOT EXISTS "Accounts" (
        "toptier_code" TEXT NOT NULL,
        "baseDataURL"  TEXT,
        PRIMARY KEY("toptier_code")
);
CREATE TABLE IF NOT EXISTS "Departments" (
        "dept_code"    TEXT,
        "toptier_code" TEXT,
        "dept_name"    TEXT NOT NULL,
        PRIMARY KEY("dept_code"),
        CONSTRAINT "FK_Accounts" FOREIGN KEY("toptier_code") REFERENCES Accounts(toptier_code)
);
CREATE TABLE IF NOT EXISTS "SubDepartments" (
        "sub_dept_code"        TEXT NOT NULL,
        "fiscal_year"  INTEGER NOT NULL,
        "dept_code"    TEXT NOT NULL,
        "sub_dept_name"        TEXT NOT NULL,
        "sub_obligated_amount" REAL NOT NULL DEFAULT 0,
        "sub_gross_outlay_amount"      REAL NOT NULL DEFAULT 0,
        CONSTRAINT "FK_Departments" FOREIGN KEY("dept_code") REFERENCES Departments(dept_code),
```

```
        PRIMARY KEY("sub_dept_code","fiscal_year")
);
```

Within the data for INSERT statements if you follow the lecture method, then an extra step of cleanup of values is needed, which for the names of department we need to look and replace a single apostrophe to a double. The name with an issue has "**Survivor's**" in it, so to replace will fix that issue. This is common for SQL interaction, so a common thing to clean up the same way. So for string data you would just use something like the following on the value: `.replace("'", "''")`

Questions:

For the following print the output to the screen to have "Question X Answer" followed by a line break and then the output. This will show the answer just after the label. Placeholders will be in the starting code block. When you see the work total, think about the aggregate function of SUM for the database query. Round amounts to 2 decimal places.

1. What is the total for the children accounts **obligated_amount** for all accounts and fiscal_years, single value only not a dataframe?
2. What is the sub_dept_code for the department that has the total amount highest for **obligated_amount** for the **020** account, single value only not a dataframe?
3. What is the dept_name for the department that has the total lowest **obligated_amount** for the **012** account, single value only not a dataframe?
4. What is the sub_dept_code and total highest **obligated_amount** for the **012** account, as a dataframe column sub_dept_code and total column name alias as sub_obligated_amount?
5. What is the department name that has the total lowest **obligated_amount** for the **012** account, as a dataframe column name to match the database column name of dept_name and total alias as sub_obligated_amount which has an amount greater than $1000?
6. Return a dataframe to include the account number (based arguments from the console: example 012 or 020), the department name (item 6 in above JSON highlight), the child sub-department code (item 7 in above highlight) and the obligated_amount (item 8 in above highlight) for the top 10 sub-departments. Name the columns in the dataframe to match the database as toptier_code, dept_name, sub_dept_code, and sub_obligate_amount. This is not to be hard coded for accounts as automated testing may pass other accounts in. Print the dataframe using the provided base code to call the function `dfToJSON`.
    a. Hint having passed 012 and 020 as the accounts: the amount should total $3,590,396,851,567.66
7. What is the total of the **obligated_amount** for the sub-department of 012-X-2278-000, which is part of account **012**, single value only not a dataframe?
    a. Hint: is it a negative value.
8. How many distinct sub-department codes are there for accounts and all years data loaded, single value only not a dataframe?

The documentation for the API is found at:

https://github.com/fedspendingtransparency/usaspending-api/blob/master/usaspending_api/api_contracts/contracts/v2/agency/toptier_code/federal_account.md

A helpful page on the site to see other accounts is https://www.usaspending.gov/federal_account. In the list, the first three digits is the account number to be passed to the API to get all sub departments of the account.

The ERD diagram was created using https://erdplus.com/.