CIS 41B - Lab assignment 3: web scraping and data storage with requests, beautifulsoup, sqlite3

Write an application that lets the user search for the top travel destinations in 2024.

The application has 2 separate modules: lab3back.py (the backend to get data)

lab3front.py (the frontend with the GUI)

The 2 modules do *not* directly work with each other (no importing of data or methods from one file to the other).

Instead: - the lab3back.py will produce a JSON file and an SQL database file

- the lab3front.py will read from the SQL database to display data to the user

**lab3back.py description**

Part A

The data are from the website: <https://www.timeout.com/things-to-do/best-places-to-travel>

* From the given URL, write code to extract the root URL of https://www/timeout.com and store it. Don’t hard code the root URL.
* From the HTML data at the URL, find the 12 month links on the webpage.
* For each month link:
  + Go to the page of the link by prepending the root URL to the month link.
  + At the page, find the travel destinations on the page (there are 10-14 of them)
  + At each travel destination, get the month name, ranking number, destination name, and description text   
    and store them in a container of your choice.  
    -- Note that each data value should be scraped from the web page. Don’t hard code any value.  
    -- [2pts EC: if the destination name is a link, also get the URL. This is step 1 of the EC.]
  + Note: for September only, you will only be able to scrape the first 10 destinations. Don’t worry about the last 4.
* Store all the data into a JSON file such that the data for one destination are grouped together and can be easily viewed in the JSON file. The goal is to make it easy to get the data back out as a group, based on destination. Therefore, storing all the data in a flat 1D list is not a good idea, for example.

Part B

After you've created the JSON file, comment out the code of Part A. You don't need to spend time fetching data from the website anymore, instead you can use the persistent data in the JSON file.

Read data in from the JSON file and use sqlite3 to create an SQL database.

* The column names can be hard coded strings. Column names are like variable names, it's your choice to name the columns, but make the name descriptive since it makes the SQL code easier to read.
* There should be 2 tables: a table of month names and a table of the all data for each destination.  
  Don’t hard code the month values in any table. The values should come from the JSON file or from the database.

Here's a partial view of a sample SQL table from the DB Browser tool:

A screenshot of a computer

Description automatically generated

[EC: Add another column for the URL of the destination, if there is a URL. This is step 2 of the EC.]

**lab3front.py description**

1. Write a GUI with 4 window classes for the 4 types of GUI windows: main window, dialog window, result window, and display window.

Make sure lab3front.py has 4 classes only.

1. The GUI *main window* has a title (your choice of text), a “Search by” string, and 3 buttons:  
    A screenshot of a computer

   Description automatically generated

* The 3 buttons let the user search for travel destinations by:
  + the name of the place
  + the month of the year
  + the rank from all 12 months

1. When the user clicks on any of the buttons of the main window:

* The main window creates A GUI *dialog window*, and the main window is no longer accessible to the user.
* The *dialog window* has:
  + An instruction to tell the user to click on a radio button. The instruction should be appropriate for the search criteria. See sample below.
  + Radio buttons that are appropriate for the search criteria, all buttons are left justified (lined up on the left). The number of radio buttons depend on the available choices.
  + The top button is pre-selected
  + Here are partial sample windows for the 3 criteria:

A screenshot of a test

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer screen

Description automatically generated

By Name By Month By Rank

* For the Name criteria:
  + Show all the first letters that are from all the destinations.
  + Make sure you get the first letters from the names in the database. Don’t hard code the letters because some letters should not be shown, if there’s no destination name associated with those letters.
* For the Month criteria:
  + Show all the months.
  + Make sure you get the months from the database. Don’t hard code the months.
* For the Rank:
  + Show all the ranks up to the max rank. Some months have fewer ranked destinations, use the max rank number of all the months.
  + Make sure you get the max rank from the ranks in the database. Don’t hard code the max.
  + When the user clicks on a radio button, the dialog window closes, and execution returns to the main window.

1. When the main window gets the user choice:
   * If the user had clicked ‘X’ to close the dialog window, then the main window does nothing.
   * If the user selected a choice, then the main window queries the database for the results, which is a list of:
     + Destination names, for Name criterion
     + Destination names, for Month criterion
     + Destination names and months, for Rank criterion

For each criterion, use only *one* SQL command to get all the results from the database.

* + Then the main window creates a *result window* and passes to it all the necessary data to display the results.  
    When the result window is displayed, the main window cannot be activated by the user.

1. The *result window* has:
   * A description appropriate for the search criteria. See samples below.
   * A listbox that can display 10 lines of text.
   * A scrollbar, since there could be more than 10 lines of data.
   * Here are 3 sample result windows from the 3 search criteria:

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

By Name, J chosen By Month, August chosen By Rank, 12 chosen

* For Name: show the names in alphabetical order. If there are duplicate names, only one is shown.  
  In the sample above only 2 names start with J, so the scroll bar has no bar for scrolling.
* For Month: show the ranks and names, in ranking order.  
  In the sample above there are more than 10 names, and the scroll bar has been scrolled down to the middle of the list, so the ranks shown are 4-13.
* For Rank: show the names and months, in month order.   
  In the sample above, not all months have 12 destinations, so only the months with at least 12 destinations are shown.
* The user can select one choice from the listbox of the result window.
* If the user clicks on a destination name to select it, then the *result window* sends 1 query to the database for the description text [and URL if doing the EC] of the destination. Then the result window opens a *display window* to display the destination.

1. The *display window* has:
   * The destination name.
   * The description paragraph.
   * Here are 2 sample display windows:

A screenshot of a blue screen

Description automatically generated A screenshot of a travel message

Description automatically generated

Example of display window with URL button Example of display window without button  
 If not doing EC, all the display windows are in this format

* + [EC: if the destination has a URL, create a button that the user can click to open a browser window to get online details of the destination from the source website. This is step 3 of the EC.]

Code to open a web browser window from a URL:  
 import webbrowser

webbrowser.open(url)

1. For the *main window* and the *display window*:
   * Add color to the background and/or the text. [Here](https://www.askpython.com/python-modules/tkinter/tkinter-colors)’s a quick list of color choices for tkinter.
   * Change the font size of the title to be larger than the rest of the text in the window.

1. Don’t use the pop-up window for confirmation when the user closes the app.  
   No need to check for file open errors.

**Documentation**

- Your name, lab number, and module name (lab3front or lab3back) at the top of each source file

- Docstring for each public method of a class. Docstring for other methods are fine, but not required.

**Turn in**

Turn in 4 separate files (not as 1 zipped file):

lab3back.py, lab3front.py, JSON file, database file