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

Write an application that lets the user search for Covid-19 data of countries of the world.

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 table of countries' Covid-19 info at the website: <https://www.worldometers.info/coronavirus/>

* From the HTML data at the URL, use beautifulsoup to find the table and extract data from each row (each country) into multiple fields.
* Real life data can be messy. If there's missing data, put in a replacement value that's equivalent to 'no data'.
* Store the fields of data into a JSON file such that it's clear which data fields belong to a country. The goal is to make it easy to get the data back out, so as an example, storing all the data fields of all countries in a flat 1D list is not a good idea.

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 any more, instead you can use the persistent data in the JSON file.

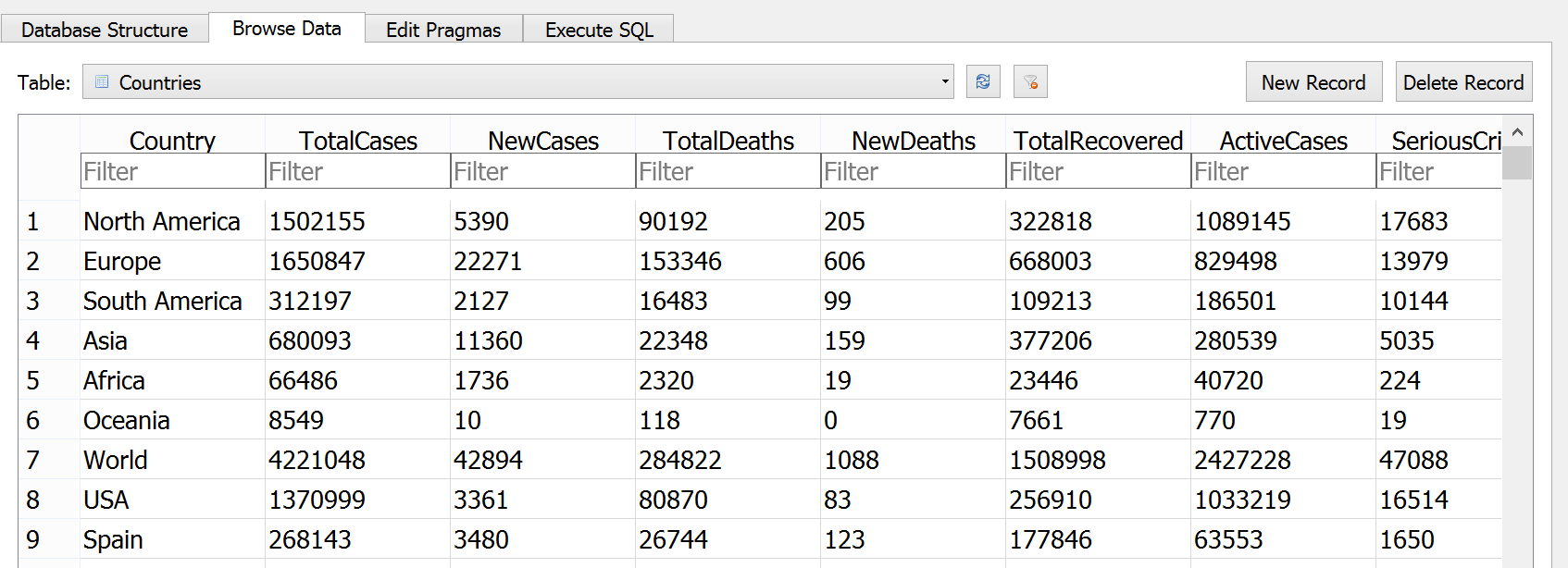
Read data in from the JSON file and use sqlite3 to create an SQL database.

* The column names can be hard coded strings. Column names are like variable names, it's your choice of name but make the name descriptive since it makes the SQL code easier to read.

TotalCases INTEGER,  
NewCases INTEGER,  
TotalDeaths INTEGER,  
ActiveCases INTEGER, ...

* The code to create the table should not be multiple lines of repeated code, such as:  
  Let Python do the repetition for you.
* Likewise, when inserting each country's data, the data should not be inserted one field at a time on a line of code.

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

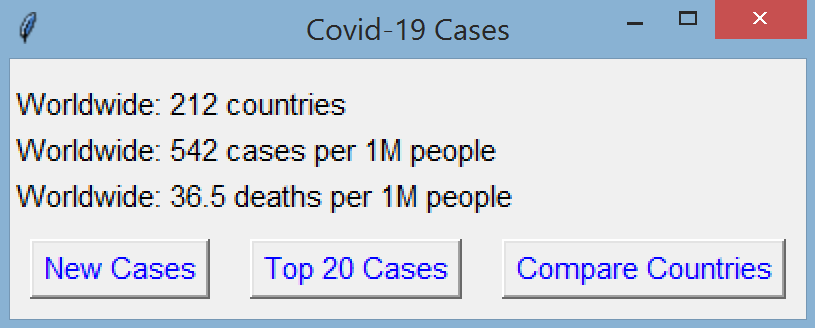


[This table was created on 5/11, so your numbers most likely will be different. The website is updated daily.]

lab3front.py description

1. Write a GUI with 4 window classes: main window, display window, dialog window, plot window.

Make sure lab43front.py has 4 classes only.

1. The *main window* shows some statistics and 3 buttons for the user to view detailed data.  
    

* The 3 lines of statistics show: the total number of countries in the database, the number of Covid-19 cases per million people worldwide, and the number of deaths per million people worldwide.
* The 3 buttons let the user see:
  + the number of new cases and new deaths for the current day
  + the 20 countries with the highest number of cases
  + a plot of the number of cases for the countries of their choice, so they can visually compare the countries.

1. When the user clicks on the button to view new cases:

* The callback function retrieves all countries with new cases from the database and sends it to the display window.
* The *display window* has:
  + A title
  + A listbox that can display 10 lines of text:
    - A header line with 3 fields: Country, New Cases, New Deaths
    - The countries are sorted in descending order by number of   
      new cases.
    - The user can click on an item in the listbox but it doesn't cause a callback function to run.
  + A scrollbar
  + A line of text that shows the highest number of new cases in a continent and the corresponding continent name



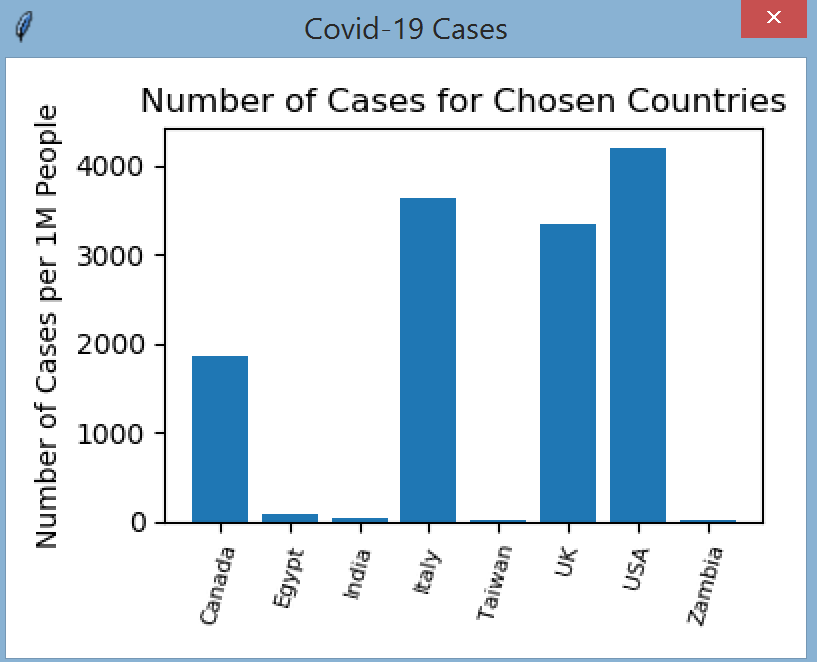


1. When the user clicks on the button to view the 20 countries with the highest number of cases:

* The callback function retrieves from the database the 20 countries with the highest number of cases per   
  1 million people, and sends it to the display window.
* The *display window* has:
  + A title
  + A listbox that can display 20 lines of text
    - A header line with 4 fields: Country, Cases / 1 million people, Deaths / 1 million people, Tests / 1 million people.
    - The countries are sorted in descending order by number of cases.
    - The user can click on an item in the listbox but it doesn't cause a callback function to run.
  + No scrollbar, since there are exactly 20 lines of data
  + And unlike the display for new cases shown above, there is no space for a line of text below the listbox.

1. When the user clicks to compare countries:

* A *dialog* *window* shows up to let the user chooses countries. The *dialog window* has:
* A listbox that can show 10 lines of text. Each line of text is a country name from the database, the list of names are sorted in alphabetical order.
* A scrollbar
* An OK button for the user to commit their choices.
* The user can click one or more country names to choose the countries, and then click OK to commit the choice and close the window.
* Back at the *main* *window*:
* If the user didn't click on any country name at the dialog window, or if the user clicked 'X' to close the dialog window, then nothing is displayed and the user is back to the main window.
* If the user made 1 or more choices, fetch from the database the data for each choice, then send the data to the plot window.
* At the *plot* *window*:
* Plot the number of cases for each country.
* There should be a title, y-axis label, and the xticks should be the country names.
* The country names are sorted in the plot
* Here's a sample plot:



Some tips on handling data to meet the requirements:

* lab3front.py should get data from the database file. Do not go to the website, or import anything from lab3back.py, or use the JSON file.
* lab3front.py should connect to the database and keep the connection open for all transactions. Don't open the database each time you need to get data. Close the database when the GUI is closed.
* Let the database do the work, don't write code to do a task if the database can do it for you. For example, you should not have to write code to calculate the average number of cases per 1 million people worldwide.
* Note that as described above, all database access are in the main window.

When done, turn in 4 files: lab3front.py, lab3back.py, the db file, and the json file.

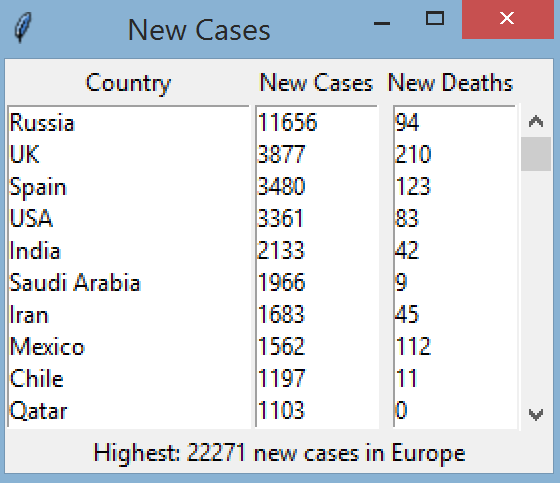
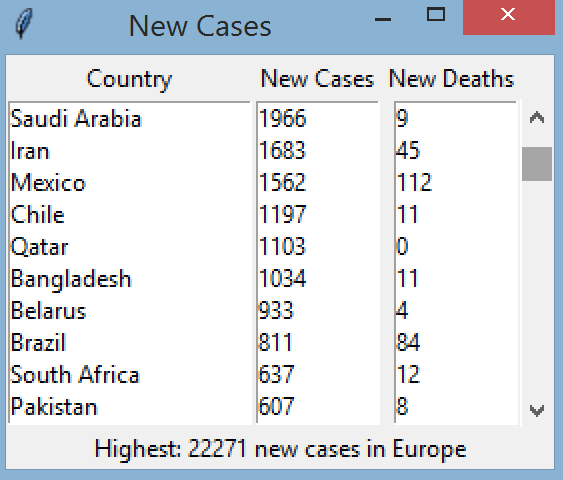
If you'd like an additional challenge, see the next page for the extra credit.

Extra credit (4pts):

The tkinter Listbox is designed to print lines of text, therefore it doesn't have an easy way to show data in column format, resulting in the ugly output of the display window shown in step 3.

For extra credit, create a new Multilistbox class that's made of multiple listboxes side by side. When data needs to be displayed in column format, each column is in a listbox, and one scrollbar will scroll all listboxes together so that the data stay together.

Here's an example of the multilistbox that displays the same data as in step e:

Multilistbox starting from the top Multilistbox after some scrolling so Saudi Arabia is at the top

The good news is that you don't have to re-invent the wheel and come up with the code for the Multilistbox from scratch. Python, just like other popular languages, have "cookbooks" of pre-written code that are commonly used. All you have to do is take the code and adjust it to your application.

The cookbook "recipe" for Mutlilistbox is at: <https://www.oreilly.com/library/view/python-cookbook/0596001673/ch09s05.html>

And a second helpful source is on Stack Overflow: <https://stackoverflow.com/questions/42715941/scrolling-through-multiple-listboxes-using-one-scrollbar-without-using-a-class>

To work on the extra credit, start with the cookbook recipe first, and then use the Stack Overflow link to supplement it. Part of the extra credit is to read someone else's code (a pro's code, in the case of the cookbook) and understand it.

To earn the 4 pts, the code must be adjusted for lab3:

* You cannot use: from tkinter import \* like in the cookbook recipe. Cookbook recipes are to show you how to do one specific thing, so the authors try to have only the code that illustrates the main task.  
  Typically you modify the recipe to fit the standards in your application, therefore, use what we've discussed in class:   
  import tkinter as tk
* Change all usage of pack() in the cookbook recipe into grid() for your Multilistbox class. If you've been using pack() in your code, here's an opportunity to learn grid(), and if you've been using grid() in your code, here's an opportunity to learn pack().
* Remove all methods that are not necessary for lab 3. This is part of understanding what the code does.
* The scrollbar in the cookbook is configured in a slightly different way than what's shown in the class notes. Change the scrollbar code of the cookbook so it's similar to the steps that are in the class notes. The StackOverflow link (first answer) shows an example that's closer to the class notes.
* Use the Multilistbox in the display window object.