CIS 41B - Lab assignment 3: web scraping, data storage, review GUI

Write an application that downloads data from a college ranking website and lets the user view the data.

The application has 2 parts: lab3back.py (the backend to get data) and lab3front.py (the frontend with the GUI)

The 2 parts do *not* work directly 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 database to present data to the user

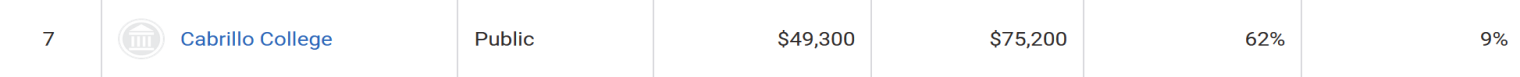
lab3back.py description

Part A: create a JSON file

1. The website that you work with ranks all two-year colleges in CA based on graduates' salary. The URL is:

<https://www.payscale.com/college-salary-report/best-schools-by-state/2-year-colleges/california/page/1>

There are 5 pages of data, and for this lab you will work with the first 2 pages. The given URL is for page 1. For page 2, substitute the '1' at the end of the URL with '2'.

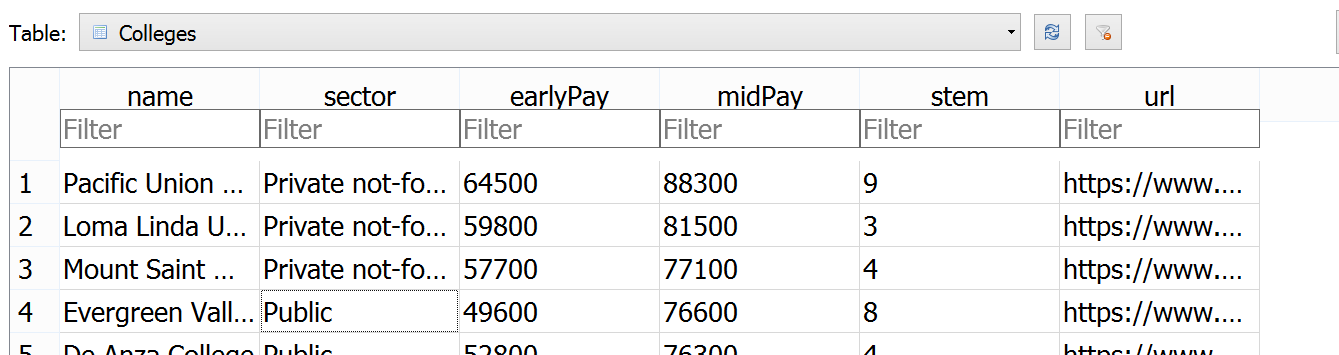
1. The data that need to be extracted are from the table that ranks the colleges based on salary potential. Each line of the table is in the format:  
      
   The fields are: rank, school name and link, school sector (public or private), graduate starting salary, graduate mid career salary, percent job satisfaction, percent STEM degrees.
2. Make the code more scalable by writing a loop that runs 2 times to extract data from pages 1 and 2. (If you had to extract all 5 pages, the loop makes the code more scalable than duplicating the code 5 times).
3. The extracted data to be saved are: school name, school link, school sector, starting salary, mid career salary, and percent STEM degrees.   
   - It would be helpful to store the salaries and percent as integers instead of text strings.  
   - Each school link is an internal link that can be accessed from within the web page. To turn it into a full link (URL) that's used from outside the web page, add the string " https://www.payscale.com" to the front of the internal link.  
   - Some schools don't have a link. It's up to you how to handle this.
4. Store all extracted data into a JSON file. You can choose the format that the data appear in the file. Choose wisely and your code can be simpler and shorter.

Part B: create a database

1. When you're done creating the JSON file, comment out all the code that you wrote for Part A. You don't need to go to the web page anymore because you can use the JSON file as input.
2. Read data from the JSON file and store into an SQLite database.
3. Each row of the database is for 1 college. The school name is the primary key, and there is a column for each of the other data fields.

You can use any order you like and any name you like for the columns that are not the primary key column.

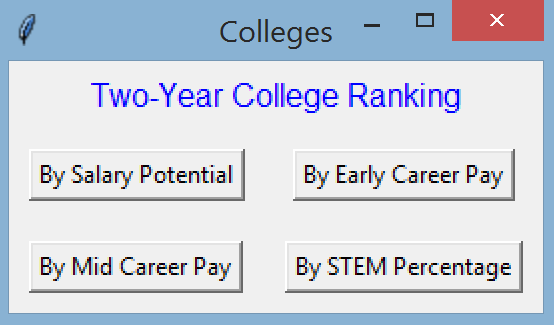
Sample of the database, from the DB Browser tool:



lab3front.py description

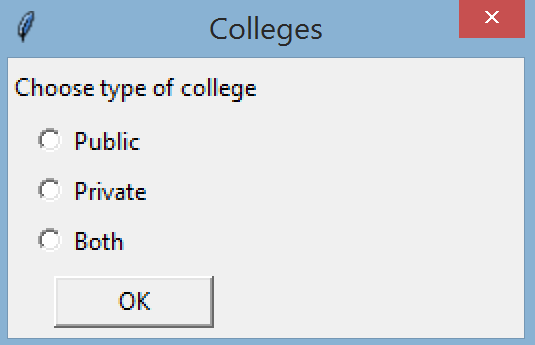
1. Write a GUI with 3 window classes: main window, choice window, and display window.
2. The main window has:

* A title, and a label to explain the app.
* 4 buttons so the user can see the colleges when ranked by salary potential (original order of the web page), by early career pay, by mid career pay, or by STEM degree percentage.
* The 4 buttons should be lined up in row and column format.
* Any font size, color, text string wording is your choice.



1. The choice window has:

* The same title as the main window
* A line of text to tell the user to make a choice
* 3 radio buttons with choices: public colleges, private colleges, or both types.
  + The radio buttons must be left justified (lined up on the left).
  + Since the buttons all do the same task (which is to record the user's choice), and the only difference is their text string, use a loop to create the 3 buttons, don't copy and paste the same code 3 times.
* An OK button below the radio buttons that the user uses to lock in their choice.

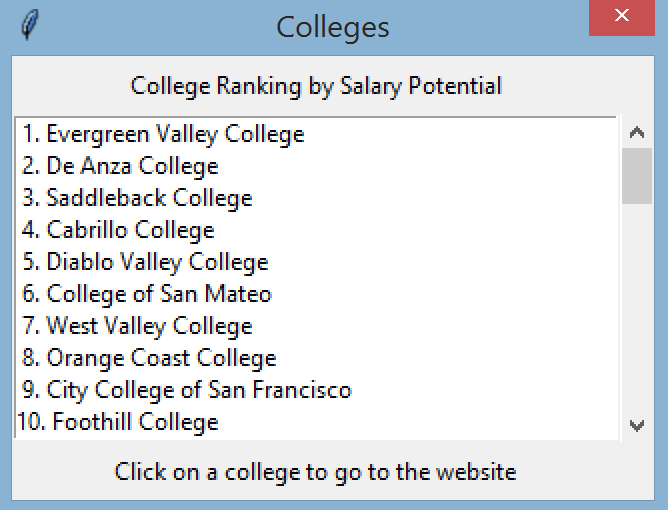


1. Behavior between the main window and choice window:

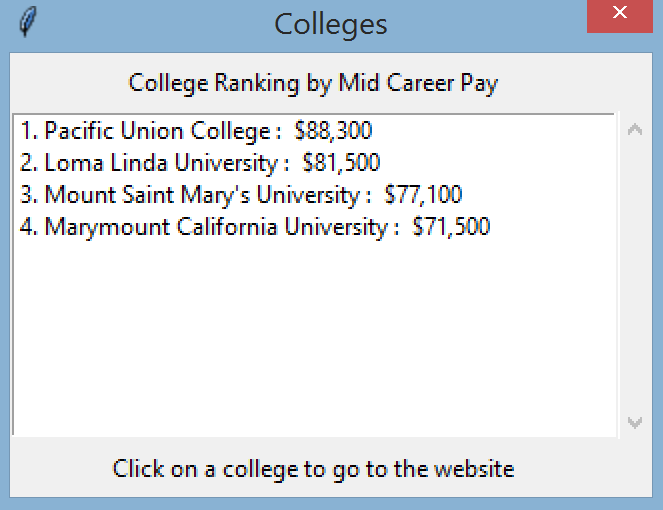
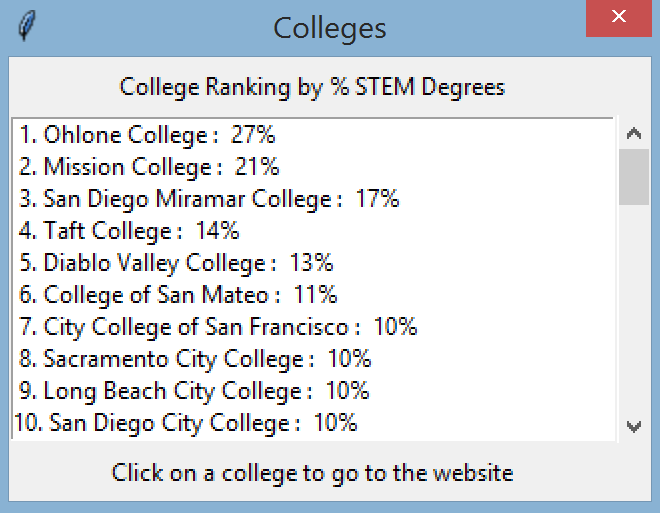
* At the main window, if the user clicks on the X, then all GUI windows close and the application ends.
* At the main window, if the user clicks 1 of the 4 buttons, then the choice window shows up.
* When the choice window shows up, all other windows are not active and the user cannot access these other windows.
* At the choice window, the user can click on the X. This closes the choice window and the user is back at the main window.
* At the choice window, the user can select a radio button and then click on the X. This closes the choice window and the user is back at the main window.
* At the choice window, the user can select a radio button and click the OK button. This closes the choice window and user choice is now available for the main window to use. The main window uses the choice to open a display window to display the appropriate college ranking (see description of the display window below).
* Note that the choice window does not create a display window.

1. The display window has:

* A title which is the same as the main window.
* A label that shows the type of ranking that the user has chosen from the main window. The wording of the label is your choice, you don't have to follow the examples below, but it should clearly show the type of ranking.
* A listbox that can show 10 items at a time, and a scrollbar for the listbox.
* A label at the bottom of the window to tell the user to click on a college name to go to the payscale website for that college.
* For the rank by salary potential: each line of the listbox has a rank (a number) and the school name, the lines are in the same order as the website.



* For the other types of ranking, each line has: the rank (a number), the college name, and the data associated with the ranking type.
  + The data associated with the ranking type must have $ and comma if it's a salary, or % if it's the STEM percentage.
  + To print a large number with comma, use the format string: f'{largeNum:,d}

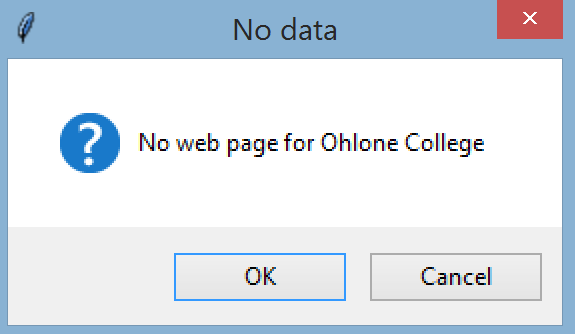
 

1. Behavior between the main window and the display window

* When the main window gets the user choice from the choice window:
* The main window queries the database for data matching both the college sector (public or private) and the category (by salaries or percentage) that the user has chosen.
* The data returned from the database should only be those that the display window will be showing. Don't fetch all fields from the database if you won't use them all.
* Then the main window creates the display window and passes the query result to the display window.
* When the display window opens, the user can still access other windows and make selection from these other windows.
* When the user clicks on a line (a college) in the lisbox, the payscale web page for the college appears as a tab in the user's browser. The display window remains open so the user can select another college if they want to.

To open a webpage: import webbrowser  
 Then: webbrowser.open('URL')

* Some colleges don't have a link. If the user selects one of these colleges, a messagebox window should appear to let the user know that there's no link, and no browser tab should be opened. The user can click any button on the messagebox to get back to the display window.



1. Some tips on handling data efficiently:

* lab3front.py should get input data from the database only. Do not go to the website or import anything from lab3back.py or use the JSON file.
* Don't fetch all data from the database to store in memory. Only fetch the data that the user requests and you need to display.
* Data is fetched from the database by the main window only. The other 2 windows are to interact with the user: the choice window is to get the user's choice, and the display window is to display the query result.

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

Optional extra credit (EC), 1pt:

For the lab3front.py, don't hard code the database column names. Instead get the column names from the database.

This makes the lab3front.py code more flexible. You are guaranteed that whoever designs the database will have the columns in the same order, but the person could change the name of the columns, and you don't want your code in lab3front.py to break if a column name changes.

To get the column names of a table: cursor.execute("PRAGMA TABLE\_INFO( table\_name )")

Then use cursor.fetchall() to see the output, which is a list of tuples. Each tuple has info on a column, and one of the values in the tuple is the column name.