CIS 41B - Lab 4: Multi-threading, multiprocessing, System, Web API

Now that the weather’s warmer and flowers are blooming, everyone’s thinking of weekend getaways and road trips…

Write an app that shows weather data for several Bay Area cities that have tourist attractions, just in time for planning the upcoming holiday weekend.

Part 1: Web access: Exploring the API

The weather data is fetched from an API of open-meteo.com: <https://open-meteo.com/en/docs>

This is a free API (as long as you make less than 10,000 API requests per day) and it doesn’t require that you register for an API key.

At the URL above, the sample GUI shows that the API needs the latitude and longitude of the city as input, and it returns the weather data for that location. Fortunately, open-meteo.com also has a geocodiing API that we can use: <https://open-meteo.com/en/docs/geocoding-api>

1. At the geocoding API, the example GUI shows that the API needs the city name, and it will return cities around the world that matches the city name. In the example, “Berlin” is the input, and the output shows “Berlin” in Germany as well as the US and Russia.
2. You can copy the example API URL on the page and change the city name in your Python code.  
   The format of a typical API request is: endpoint?param1=value1&param2=value2&param3=value3 …

where: the endpoint is the start of each API request, ending with ?

the criteria are separated by & and each criterion tells the API what you want

In the example API request , the criteria are:

- look for the city named “Berlin”

- want to see up to 10 cities with that name

- the language is English

- the return data format is JSON

See the API Documentation section for more explanation on each criterion or parameter.

1. The returned JSON data has multiple fields, including the latitude and longitude that you will need.
2. For Lab 4, there are 10 cities that you will work with:

North Bay : Napa, Sonoma

The Coast: Santa Cruz, Monterey

East Bay: Berkeley, Livermore

Peninsula: San Francisco, San Mateo

South Bay: San Jose, Los Gatos

1. Make the API requests for these cities, then look at their JSON returned data and write code that will choose the correct cities that are in the Bay Area so you can get the correct latitude and longitude.
2. After you’ve got all the geocoding data for all 10 cities, borrow the trick from Lab 3 and save the data to a file so you don’t have to do 10 API requests each time you run your app to work on your lab (we’re trying to stay below 10,000 requests/day 😉).
3. After you have the longitude and latitude of all 10 cities, you can look up their weather with the weather forecast API of open-meteo.com shown above. At the weather forecast API page, the example GUI shows that the API needs the latitude and longitude as input.

Look up the API Documentation and choose the following parameters:  
- *daily* weather data for 5 days, starting from the current day that the user runs the app and the next 4 days

- dates for the 5 days

- max and min temperatures (at 2m above ground)

- max wind speed

- max UV index (how sunny it will be)

The temperature should be in Fahrenheit, the wind speed should be in mph, and the time should be local time.

Hint: it’s easier to work with just one city until you get all the needed data for that city. After the code works with one city, all the other cities will be the same, that’s the nice part about working with an API, the data is much more consistent than when web scraping.

Part 2: GUI and Systems

After your code can fetch all required data from the API, add the GUI component.

The application has 2 classes for the 2 GUI windows: a main window and a display window.

1. The application starts with a main window.

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | The window has:   * a title * the purpose of the app * an explanation for how to use the app * a listbox with the 5 areas and 10 cities (you can hard code these strings) * a button   The user can click on the listbox items and choose one or more destinations, then click the Submit button to get results for their city choices. |

1. To populate the listbox with the 10 cities, the app checks to see if there’s already an input file (your choice of filename).

- If there is, it reads in the city geocodes from the input file.

- If there isn’t an input file, it makes the API requests to fetch the geocoding data to use in the app and also to save to the input file.

1. The user selects and submits their choices.

|  |  |
| --- | --- |
|  | * The user can click to select or unselect their choices as many times as they like. * When the user clicks the Submit button, then the main window makes one API request for each chosen city. * When data for all the chosen cities are fetched:   + the user selections are cleared from the listbox (so the user can choose again without having to unselect the previous choices)   + for each city, the main window creates a Display window to display the data. |

1. The Display window shows the 5-day weather data for one city.

|  |  |
| --- | --- |
|  | The window has:   * A title * A header line with the city name * 5 listboxes to display the 5 types of data:   + Dates, starting from the current date   + High temperature   + Low temperature   + Max wind speed   + UV index * Each listbox has a header that centers above it to explain what the data are. Note that you should not have one label with all 5 headers, since it’s a lot of work to center each one above its listbox, *and* the centering might not work when the window is displayed on a computer that has different screen resolution than yours. |

1. After the data for the chosen cities are displayed, the user can go back to the main window and choose the same or new cities to see more weather data.
2. When the user clicks X to close the app, if the user has looked up weather data for at least 1 city, then a messagebox window pops up to ask if the user wants to save all their search results.

A screenshot of a computer

Description automatically generated

* + - The user can click OK to save or click Cancel to not save and quit out of the application.
    - If the user clicks OK, a directory select window shows up and displays the current directory where the app is running. From this window, the user can navigate to different directories and choose a directory.
    - If the user clicks 'X' to close the directory select window, then no file is saved and the application ends.
    - If user chooses a directory, then file "weather.txt" is created in the chosen directory.
    - The weather.txt file does not contain duplicate data for a city.

For example, if the user selects San Jose and San Mateo and see the results, then selects San Mateo and Napa and see the results, then weather.txt has data for 3 cities: San Jose, San Mateo, and Napa.

* + - For each city in weather.txt, data are in the following format:

San Jose:

2024-05-20, 2024-05-21, 2024-05-22, 2024-05-23, 2024-05-24

76.4, 81.8, 79.7, 75.9, 79.6

48.0, 50.7, 51.4, 52.3, 49.6

10.6, 13.1, 13.5, 14.5, 15.4

7.9, 7.95, 8.0, 8.0, 7.9

* + - When weather.txt has been saved, a pop up window reminds the user where the output file is:

A white rectangular object with black text

Description automatically generated

The app ends when the user selects OK to acknowledge.

Part 3: Multithreading

There are 2 bottlenecks in the app:

- the 10 API requests to get the geocoding for the cities

- the 2-10 API requests when the user wants to see weather data for one or more cities

To observe the bottleneck:

- Remove the input file of geocodes so that the code is forced to make the 10 API requests upon GUI start up.

- You’ll notice that the GUI doesn’t respond right away.

Rather than making multiple API requests serially, create a thread for each API request so they can run more in parallel.

1. Make a copy of the code that makes multiple API requests serially in Part 2.
2. Modify the copy so it makes the requests with one thread per request.
3. Run the API requests serially and measure the elapsed time. Make sure you measure the data fetching time only.
4. Run the API requests with threads and measure the elapsed time. Make sure you measure the data fetching time only.
5. At the end of your source file, create a comment block to show the 2 different elapsed times.  
   The format of the comment block is:

serial multithreading

geocoding data N1 N2

weather data N3 N4 where N1, N2, N3, N4 are measured and printed from your code

EC (2pts):

For the 10 API requests for geocoding data, use any multithreading methods we’ve covered in class to make the elapsed time as short as possible.

Part 4: Multiprocessing

1. Make a copy of your .py file with threads and use the copy for processes.
2. Change the multithreading for API calls into multiprocessing for API calls:
   * + - For the 10 API requests for geocoding data, use a multiprocessing pool.
       - For the API requests for weather data, use a multiprocessing queue.
3. Change the 2 methods to request to data from the API into 2 global functions so that they can be used by the child processes.
4. Run the API requests with processes and measure the elapsed time. Make sure you measure the data fetching time only.
5. Add to the time measurement table you have in the lab 4 threading file:

serial multithreading multiprocessing

geocoding data N1 N2 N5

weather data N3 N4 N6

- Rank the 3 ways to make the API requests (series, multithreading, multiprocessing) in order from slowest to fastest time.

- Explain why they are in that order.

Documentation

- Your name, lab number, module name at the top of each file

- Docstring for each public method, docstrings for private methods are good but not required.

Turn in:

lab4thread.py and lab4process.py