CIS 41B - Lab 4: web api, multithreading, multiprocessing, os, review of json, gui

Since summer vacation is right around the corner (if not already here) for many students, write an application that lets the users look up information on US national parks so they can plan their summer vacation trips.

The lab consists of 2 files: lab4thread.py and lab4process.py. Both files run independently of each other and both files do the same work and provide the same results, except one file uses multithreading and the other file uses multiprocessing.

Follow the steps below in the listed order to build up your lab4thread.py file first. Then convert the multithreading part to multiprocessing to produce lab4process.py.

Step 1: web access

The data for all US national parks is from the National Park Service (NPS), and the NPS has provided an API for developers to easily access park data.

A. Getting started

To use the API, you need to register for an API key. Go to: https://www.nps.gov/subjects/developer/get-started.htm to register and receive a key (a long sequence of alphanumeric characters) through email.
Then click on the API Guides page link at the end of the same Get Started page to read about how to protect your personal API key, and to see how to use your API key in a request.

B. Fetch data

At the same Get Started page, right under the "Signup" button, is the description for the 3 parts of the API request:
- Resource endpoint, which is always the same
- Query string, which changes with each request, depending on what data you're requesting
- Your API key, which is always the same

Click on the API Documentation link at the end of the same Get Started page to see a list of possible query options.
The option you will use is parks or, in the request string: /parks
Click on the /parks to see the parameters you could use in the query, and to see example data that you would receive.
Using /parks in the query string and filling in the parameter for state code, you can get data for all parks in a particular state. *The data for all parks in a state is what you'll use for the lab*.

To see examples of using the query in code, NPS also has sample code on how to request certain data, which you can find at NPS on GitHub at the menu at the top of the page.

C. Get the state code

The state code in the request string is the 2-letter state abbreviation (CA, TX…). Continuing the theme of finding your own data, github can be a good source for basic datasets. Go to https://gist.github.com/mshafrir/2646763 to download and use the state_hash JSON file when you need to find the state names or state codes.

Step 2: lab4.py

Before diving into multithreading and multiprocessing, it's best to get the lab to work in a single thread first.

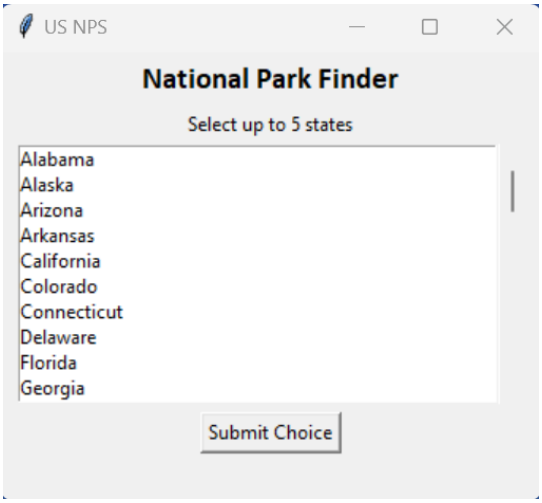
A. Overview

Lab4.py has one GUI window class which is a Tk window.
The window does 3 tasks one after another:
- present the user with a list of states so the user can choose up to 5 states

- for all states that the user chooses, present the user with a list of parks in the states so the user can choose their parks
- save data of the chosen parks of each state in a file that's the state name

B. Details

1. Let the user choose 1-5 states



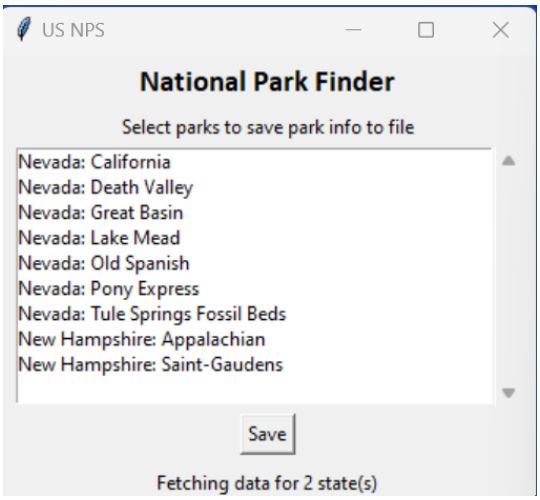
The window has these components:

- A title for the window, such as NPS or Parks, etc.
- A name for the app. Feel free to use your own wording, font type, color, size ,etc.
- A prompt that asks the user to select up to 5 states from the listbox. Make the 5 a *class attribute* so it can be changed easily.
- The listbox can display 10 items at a time and is 40-50 chars in width. It has a scrollbar so the user can scroll through all 50 states. The state names come from the states_hash.json file.
- The user clicks the Submit button to lock in the choice of states. Feel free to change the text, font, color for the button.
- A blank label at the bottom for status messages later.

2. When the user clicks the Submit button, check that the user has made 1 - 5 choices, inclusive, from the listbox. If there are not 1-5 choices, pop up a messagebox to display an error message, then wait for the user to choose again.

3. When the user has submitted 1-5 choices, make the API call for each choice, and save the Response data of all the chosen states in a container of your choice.

4. Let the user choose their parks from your data container.



Some of the window widgets change to show the parks:

- The prompt now asks the user to select one or more parks.
- The listbox shows all the parks for each state that the user chose. The format is: state name: park name
  The park name is the "name" field of the NPS JSON structure.
- The button text changes to let the user know it's for saving data to file.
- The label shows that 2 states were chosen.

(Hint while debugging: the 2 example states only have a few parks, so you can easily view both in the listbox)

5. When the user clicks the Save button, check that there is at least 1 choice from the listbox.
   If not, pop up a messagebox to display a "no park chosen" message, then wait for the user to choose again.

6. If the user makes at least 1 choice, pop up a file dialog window to let the user choose a directory to save the output file. The default directory of the file dialog window is the user's current directory.

7. If the user doesn't choose a directory, clear the user's selection from the listbox and wait for the user to choose their parks again.
   To clear the listbox, use: LB.selection_clear(0, tk.END) where LB is the listbox object.

8. If the user chooses a directory, create a JSON file for each state that the user chose:
   - The name of the file is the state name.
   - In each JSON file is a list of the parks that the user chose.
   - Each park is a dictionary with the following fields (from the NPS JSON file):
     - full name
     - description
     - activities (as a comma separated list)
     - url
   - Example JSON file of 2 parks selected from Nevada above:

```
[
    {
        "Lake Mead": {
            "full name": "Lake Mead National Recreation Area",
            "description": "Swim, boat, hike, cycle, camp and fish at Am
            "activities": "Auto and ATV, Scenic Driving, Biking, Road Bi
            "url": "https://www.nps.gov/lake/index.htm"
        }
    },
    {
        "Pony Express": {
            "full name": "Pony Express National Historic Trail",
            "description": "It is hard to believe that young men once ro
            "activities": "Auto and ATV, Scenic Driving, Guided Tours, S
            "url": "https://www.nps.gov/poex/index.htm"
        }
    }
]
```

9. After the file(s) are written, show a confirmation messagebox to let the user know the filename(s).

Confirmation messagebox after the user chose parks from Nevada and New Hampshire.

10. When the user clicks OK on the messagebox or clicks to close the messagebox, the app ends.

11. Add code to record the time it takes to get the data from the NPS API, for 3-5 of states. Be careful where you put the timer code so that the time is only for downloading the data from the NPS.

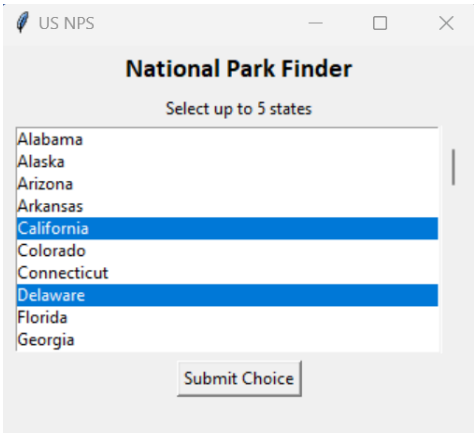
NOTE: while working with the listbox, there are 2 more listbox methods that could be useful, in addition to the ones already in the class notes: LB.selection_clear(0, tk.END) to clear the user selections in the listbox
LB.get(0, tk.END) to get all the items that are shown in the listbox

Step 3. lab4thread.py

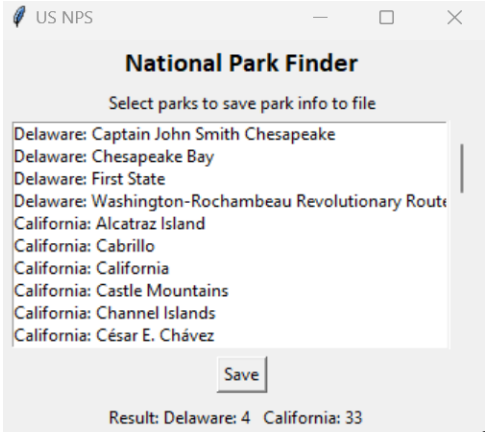
After lab4.py works to specs, make a copy of it and name it *lab4thread.py*, then change the code to use multithreading.

1. For each API call in step 3 of lab4.py above:
   - Create a thread to request data from the API.
   - Save the Response data in the same container as lab4.py

2. As each thread finished fetching the data, use the label at the bottom of the window to display the state name and the number of parks there are in the state.
   - The number of parks is the "total" field in the NPS JSON structure.
   - Note that the status message for each state should show up *as each thread is finished*, don't print all the status messages at the very end when all threads are done.



The user chose California and Delaware, and because California appears before Delaware in the list of states, the thread for California started before the thread for Delaware.



Yet Delaware status appears first because Delaware has fewer parks and can finish faster than California.

3. Add code to record the time it takes to get the data from the NPS API, for the same states that you used in lab4.py. Be careful where you put the timer code so that the time is for downloading the data from the NPS.

Step 4: lab4process.py

Make a copy of lab4thread.py file and save it as lab4process.py, then change the code to use multiprocessing.

1. Change all multithreading code to multiprocessing code:
   - Remove the multithreading code and replace with a multiprocessing Pool.
   - To minimize the code dissection and keep the later part of the app the same, it would be best to store all the Response data in the same container that was used for lab4.py and lab4threading.py.
2. For the window, the label at the bottom is now the same "Fetching data for N states" as with lab4.py.
3. The function that the processes will run should be removed from the window class and turned into a global function so that the process can access it.
4. Add code to record the time it takes to get the data from the NPS API, for the same states that you used in lab4.py and lab4threading.py. Be careful where you put the timer code so that the time is for downloading the data from the NPS.

Step 5: Analysis

At the end of lab4process.py, add a comment block to discuss your time measurements:
- Show the order from slowest to fastest between the 3 ways to fetch data: serial, threads, processes.
- Explain why they are in that order.

When done, submit lab4thread.py and lab4process.py