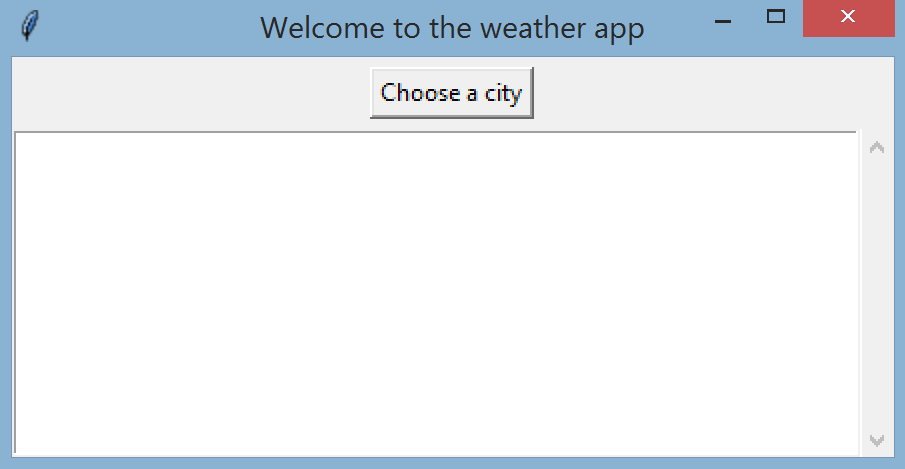
CIS 41B - Lab 4

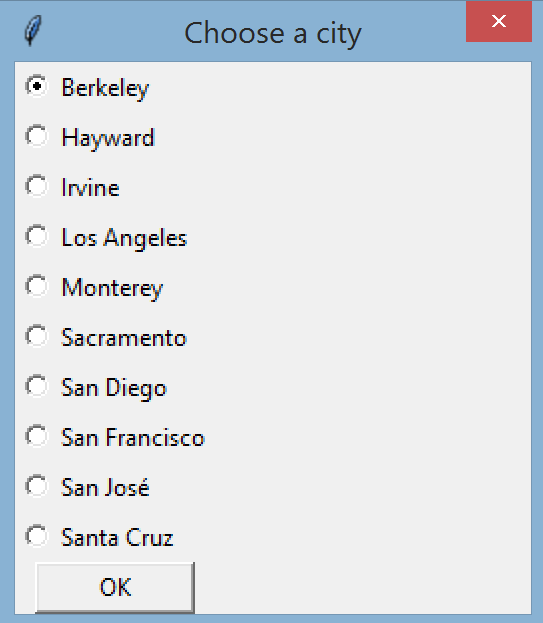
Write a weather look up application that lets a student check the temperature and weather condition of California cities with popular public universities.

Part 1: GUI and System

The application starts with a GUI window that contains a button and a listbox.

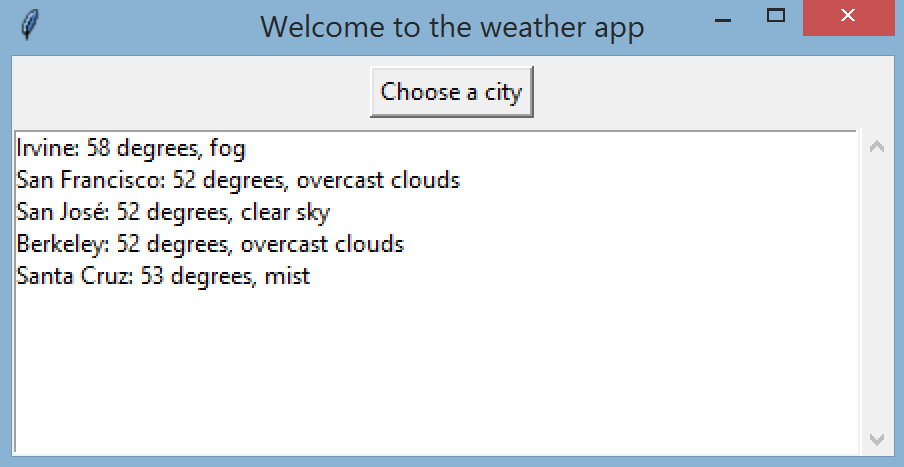


When the user clicks Choose a city, a dialog window shows the city choices in alphabetical order.

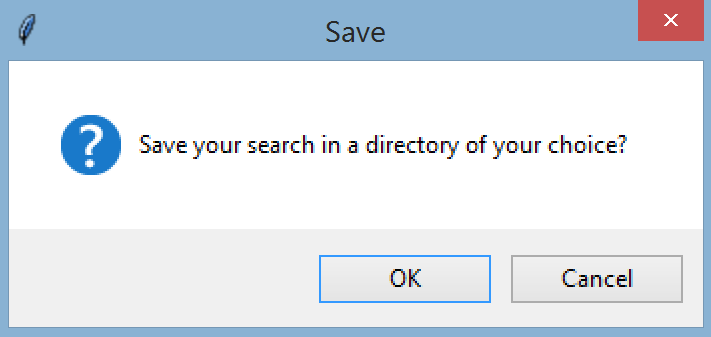


When the user selects a city and clicks OK, the current temperature and weather condition show up as one line of text in the listbox. As the user selects more cities, more lines of text are added to the listbox.

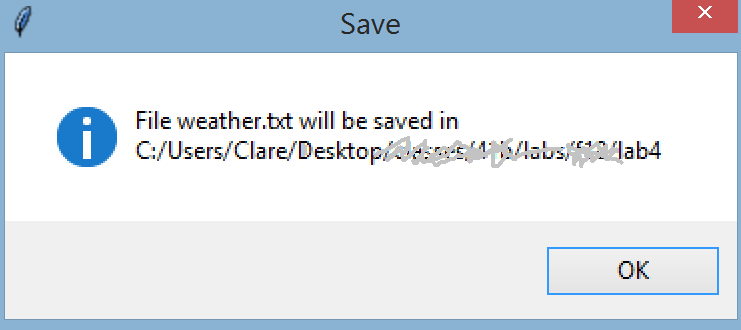
Each line of text contains the city name, the current temperature (in Fahrenheit, rounded to the nearest whole number), and a description of the current weather condition.



When the user clicks X to close the main window, if the user has made at least one selection in the listbox, then a messagebox window pops up to ask if the user wants to save their search results. They can click OK to save or click Cancel to quit.



If the user selects OK, a directory select window shows up, displaying the current directory where the python file is running. The user can use the window to get to different directories and choose a directory. When the directory select window closes, the file "weather.txt" is created in the chosen directory. Then a messagebox window appears with a confirmation message, showing the full path of the chosen directory and the new filename.



Part 2: Web access

The current weather data is fetched from an API of OpenWeatherMap.org: <https://openweathermap.org/current>

Scroll down to the heading "By ZIP code" and you will see the orange color API call, which makes up the example URL:

http: api.openweathermap.org/data/2.5/weather?zip=94040,us

In the example above, the zip is 94040 (Mountain View). With the URL you also need to append 2 more strings:

1. the string: &units=imperial so the temperature will be in Fahrenheit instead of the default Kelvin temperature

2. your key, which is a multi-digit hexadecimal string, such as: &APPID=23d3a44948b47338224e4904

The complete URL in your code would look something like this:

http: api.openweathermap.org/data/2.5/weather?zip=94040,us&units=imperial&APPID=23d3a44948b47338224e4904

To obtain a key, click on the "sign up" link of the OpenWeatherMap link above. You need a log in id, an email address, and a password. You only sign up for the free account, which limits you to 60 data fetches per minute maximum (which should be more than enough to complete this lab).

For the cities used in this lab, the list of zip codes is:

[92093, 90013, 95192, 94132, 94720, 95064, 95819, 92697, 93940, 94544]

which you can store as a class variable or a global variable..

As the OpenWeatherMap website states, the data you download from the API is JSON format.

Each data record is fairly large, with nested structures, so it is useful to download it to a JSON file with indentation so it's easier to see the relationship between the data values (dictionary of dictionaries or dictionary of list of dictionary)

Out of each data record you need 3 data values: the city 'name', the 'temp', and the weather condition 'description'  
All quoted strings are fields that you should look for.

The city names that are displayed in the dialog window are extracted from the API downloaded data. Don't hard code the city names yourself

Part 3: Multithreading and multiprocessing

In lab 3 you had the 'luxury' of downloading the data ahead of time to a file, which means that the GUI can quickly read in all the data from the file and be ready for the user. In this lab, the weather data are updated often enough that you need to fetch new data each time that the user uses the application.

There are 10 calls to the API to fetch data for the 10 cities, and they generally take about the 2 minutes maximum time that the user has patience for.

Step 1: fetch the data for the 10 cities in series and record how long it takes, using time.time(). Make sure to start the

timer right before the first data fetch and stop right after the last data fetch.

Step 2: fetch the same data using multithreading and record how long it takes.

Step 3: fetch the same data using multiprocessing and record how long it takes.

At the end of the source file, in a comment block, discuss your finding and explain which of the 3 ways you think is best.

Your explanation should discuss the pros and cons of each of the 3 ways, and then your choice for the best way.

Implementation detail for part 3:

1. Because we want the GUI to appear to be more responsive to the user, do the following steps in the order below for the main window constructor:

- set up the window with all the widgets

- run: self.update() which will make the window appear on screen with all the widgets

- continue with code to fetch data and initialize the rest of the instance variables

Because the window appears before we start fetching the data, it looks as if the application is ready for the user (even though data will still be downloading).

2. Create 2 copies of your lab 4 and call them lab4thread.py and lab4process.py.

This way you can easily compare and contrast how they are different in code and with run time.

Turn in:

lab4thread.py and lab4process.py