Name: Evan Leglar	**Lab Time** Tuesday 12:00 pm

Names of people you worked with:
• N/A

Websites you used:
https://stackoverflow.com/questions/12353288/getting-values-from-json-using-python https://carbon-intensity.github.io/api-definitions/?python#get-intensity-date-date lots of youtube… python3 JSON get data from sublevels

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	12+

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Evan Leglar

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

BEFORE YOU BEGIN

1) Read up on JSON (JavaScript Object Notation). JSON is a data format which can communicate data as text between different clients, and is commonly used as means of communication between web interfaces and various programming languages, e.g. if a program in Python wants to retrieve some data from a website.
2) Python has a number of very useful JSON libraries; look up ones which may be useful for you. You will not actually need to worry about the specifics of how the JSON is formatted or parsed or anything like that, because someone out there has already done it for you.
3) Download the test data file **carbon_2019-10-31.json** available on Canvas. Try to use this data as much as possible instead of querying the internet (i.e. you should be able to do most of this assignment without actually having your computer connected to the internet).

Learning Objectives:
You should be able to do the following:

- Format Unix timestamps to be human-readable
- Query web APIs and cache the corresponding data
- Utilize data retrieved from JSON structures

Thoughts (come back and read these after you've read the problems):

1. Making an API call to the carbon database is essentially the same as navigating to a URL in your browser with a certain format. See if you can figure out how to get a JSON response in your browser first, and then apply that to your Python code.

2. `query_carbon` is a very robust function with a lot of functionality. Take each step one at a time! Don't forget that you can prevent your functions from becoming unwieldy by separating out certain logic into separate functions.

3. As noted later, **please don't query the API super quickly!** If you make too many requests at once, the API may mark your computer as a spammer and block you. As a rule of thumb, each time you run your code, it should make at most 1 fresh API call.

Grading Checkpoints

1. `get_current_day` works **[2 points]*** and outputs the current date when no arguments are passed in **[1 point]***.

2. `query_carbon` successfully connects to carbon API given a date string **[1 point]*** and returns the resulting data as a Python dictionary **[2 points]***. It should return today's data when given no date string **[1 point]***.

3. `query_carbon` raises an Exception if the server responds with anything other than a 200 response **[1 point]^**

4. `query_carbon` saves a file with the appropriate name/location **[1 point]*** which consists of valid JSON **[2 points]****

5. `query_carbon` implements a use_cache argument when, when set to True, checks to see if the corresponding JSON file exists and loads it in if it does **[2 point]****. If it doesn't or if use_cache is set to False, it queries the API for new data **[2 points]^**.

6. `plot_carbon` outputs a PNG file with the correct name in the correct location **[1 point]*** with an appropriate title and appropriately labelled axes **[2 point]**, and uses the current date when run without arguments **[1 point]^**.

7. Code does not output any text or run any code on import. **[1 point]**

8. **Extra Credit:** `get_current_day` accepts an integer utc argument which returns the date in the corresponding time zone **[1.5 points]^**

9. **Extra Credit:** `query_carbon` checks if a data file loaded from cache has any null data and re-queries the API if it does. **[1.5 points]^**

* Code will be autograded and you will receive immediate feedback on whether it's correct.
** Code will be checked to make sure that it can be run, but will not give feedback on whether the output is correct or not.
^ Code will be automatically graded, but only shown after grades are published.
No marker means it will be graded manually.

Hand in the files and this PDF to Gradescope.

Problem 1 Setup

Create a file called `carbon.py`, and **write a function called `get_current_day`** which takes in a single argument, an integer Unix timestamp (i.e. the number of seconds since Jan 01, 1970), and returns the corresponding day on your computer in the format "YYYY-MM-DD". **We should be able to call the function without passing in any arguments**, in which case, your function should return the current day.

Tips:

1. You can get the current Unix timestamp by using the `time` function in the `time` module.
2. The format listed is called ISO 8601 format. There are standard library functions which return the desired formatting.

Extra credit: Modify your function to take a second optional argument `utc` which takes an integer specifying the time zone offset from GMT, and outputs the ISO 8601 date for the corresponding region. You can assume that DST is not active.

Comments for grader/additional information (if any)

Problem 2 Data querying

The data source we will be using is the Carbon Intensity API, which publishes data on carbon intensity forecasts and realizations in Great Britain. You can find the API documentation here: https://carbon-intensity.github.io/api-definitions/#carbon-intensity-api-v2-0-0

Please do not make extremely frequent calls to this API! (E.g. more than once every ten seconds.) If you accidentally make 100 requests to the API in a second, it's possible for the API to think you're a spammer and block your client from connecting. **Try to work with cached data for as much of this problem as you can.**

Click on "Carbon Intensity – National", and select the "GET /intensity/date/{date}" option. This will give you some information on how to use the API, including some Python code that queries for the desired data.

For this section, **write a function called `query_carbon`** which takes two optional arguments: an ISO 8601-format date string, and a Boolean argument `use_cache` which defaults to `True`. `query_carbon` should do the following:

1. It should send a request to the appropriate URL and check to make sure the status code of the response is 200. If it is something else, throw an Exception.
2. It should return the dictionary of the response which contains the desired carbon data for the given date.
3. If no date is given, it should retrieve the forecasts for the current day.

`query_carbon` should also implement **caching logic**, as described here:

1. When we make a request for data on a given date, we should take the response and output it in JSON format to the file "data/carbon_{date}.json", e.g. if we query for the date 2018-05-15, we will have a file "data/carbon_2018-05-15.json". (The outputting to a file is in addition to returning the Python dictionary at the end.)

   Note: You can assume the data folder already exists on your computer, so your code doesn't have to handle creating it if it doesn't exist.

2. When we call `query_carbon` for a given date, it should check to see if there is a corresponding file **before** attempting to query the API; if there's a file, it should load the data from the file and **not** query the API.
3. This all assumes `use_cache` is equal to `True`. If `use_cache` is set to `False`, you should ignore any cache files, query the API as before, and then create a new cache file with the newly-retrieved data.

Tip: We highly recommend writing your caching logic first by using the test data file **carbon_2019-10-31.json** provided on Canvas.

Extra credit: When `use_cache` is equal to `True` and retrieves data from a cached JSON file, check to see if the dictionary contains any null data. If it does, ignore the cache file and query the API for updated data, and then override the current cache file.

Comments for grader/additional information (if any)

Problem 3 Plotting

Write a function called `plot_carbon` which takes in a single optional argument, an ISO 8601-format date string (it should default to using the current day if no argument is passed in), and outputs a plot of the predicted and realized carbon intensities for that date. Your plot should meet the following specifications:

1. It should have a title with the date, labelled axes, and a legend for each line with clearly distinguishable line styles.
2. Each carbon intensity value should be associated with a float from 0 (inclusive) to 24 (noninclusive) representing the time from start of the forecast period. You can assume that the data starts at midnight and increases in half hour increments.

   Note: Due to weirdness from DST, if you query for a date during DST (e.g. 2019-08-20), the forecasts will actually start from 11 PM the night before. Don't worry too much about this; for now, you can assume that the data actually starts at midnight.
3. It should automatically create a PNG image file called "plots/carbon_{date}.png". Again, you can assume the plots folder exists in your working directory.

Output a plot for the date 2020-01-15 and paste the image into this document.

Comments for grader/additional information (if any)
Tried for couple hours (6+) to get the intensity data out from sublevels of the JSON dictionaries but no luck. Maybe in the future, students could get some guidance or direction to a resource on how to get to the key&value they need. Couldn't finish rest of assignment.

carbon_2020-01-15.png
Paste your image here