Before you begin

- 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.
- Python has a number of very useful JSON libraries, in particular one simply called json. You will not actually need to worry about the specifics of how the JSON is formatted or parsed or anything like that, because the json library has done it for you. You should, however, understand the structure of the data being represented in the JSON file.
- 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):

- 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.
- *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.
- 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 Rubric: [22 points]

In regards each item, please see associated footnotes for each item * † ‡

- Problem 1, setup
  - [2 points] *get_current_day* works *
  - [1 points] *get_current_day* outputs the current date when no arguments are passed *
- Problem 2, data querying
  - [1 points] *query_carbon* successfully connects to carbon API given a date string *
  - [2 points] *query_carbon* returns the resulting data as a Python dictionary *
  - [1 points] *query_carbon* returns today's data when given no date string *
  - [1 points] *query_carbon* raises an exception if the server responds with anything other than a 200 response ‡
  - [1 points] *query_carbon* saves a file with the appropriate name/location *

*Indicates that the autograder will tell you if this problem is correct on our set of test cases and assign you credit.
†Indicates that the autograder will verify that your data is of the right data type and that it is within 1 log2 order of magnitude; however, it will not tell you if your answer is actually correct. Full points will be assigned for getting within 1% of the correct answer. Half points will be assigned for getting within 10% of the correct answer.
‡Indicates the autograder will grade this in the background, but only tell the result after the homework has been published.

  - [2 points] File consists of valid JSON †
  - [2.5 points] *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 †
  - [2.5 points] If it doesn't or if *use_cache* is set to False, it queries the API for new data ‡

- Problem 3, plotting
  - [1 points] *plot_carbon* outputs a PNG file with the correct name in the correct location *
  - [2 points] Plot has an appropriate title and appropriately labeled axes
  - [1 points] *plot_carbon* uses the current date when run without arguments ‡

- Standard Grading Checkpoints
  - [2 points] Code passes PEP8 checks with 10 errors max *

- Extra Credit
  - [2 points] *query_carbon* checks if a data file loaded from cache has any null data and re-queries the API if it does ‡

Problem 1: Setup

Create a file called *carbon.py* and write a function called *get_current_day*. Your function should take in a single argument, an integer Unix timestamp (i.e. the number of seconds since Jan 01, 1970), and return the corresponding day in UTC time (i.e. without considering any time zones or DST), 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 time for the current timestamp.

Tips:

- You can get the current Unix timestamp by using the time function in the *time* module.
- The format listed is called ISO 8601 format. There are standard library functions which return the desired formatting, particularly in the *datetime* module. (You may be interested in the *utcfromtimestamp()* method in particular.)
- There are many utilities online which will do timestamp conversion, such as this one. You should make sure your answer is consistent with these utilities.

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.

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:

- 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.
- It should return the dictionary of the response which contains the desired carbon data for the given date.
- If no date is given, it should retrieve the forecasts for the current day.

query_carbon should also implement caching logic, which is described here:

- 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.
- 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.
- 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.

Tips:

- We highly recommend writing your caching logic first by using the test data file *carbon_2019-10-31.json* provided on Canvas.
- If you Google for stuff involving caching, you will likely find lots of packages on things like memoization and other things. However, those are completely unnecessary. If you just follow the logic mentioned above, you can (and should) implement the caching system using nothing more than standard file-writing utilities and the *json* module.

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.

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:

- It should have a title with the date, labelled axes, and a legend for each line with clearly distinguishable line styles.
- 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. 2021-08-20), 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.
- 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 2022-04-15 and include the image in your zipped folder with your code.