Web Requests

Computer Science - Week 10  
Oct 30, 2022 - Version 0.1.2

Please make sure that all members of the group place their UD **email** AND **name** below.

Choose roles following the [instructions here](https://blockpy.cis.udel.edu/assignments/reading/bakery_appendix_pogil).

You should work in groups of 3. If you cannot find 3 group members, then work in groups of 2.

| **Role** | **Name** | **Email** |
| --- | --- | --- |
| **Manager** |  |  |
| **Speaker** |  |  |
| **Recorder** |  |  |

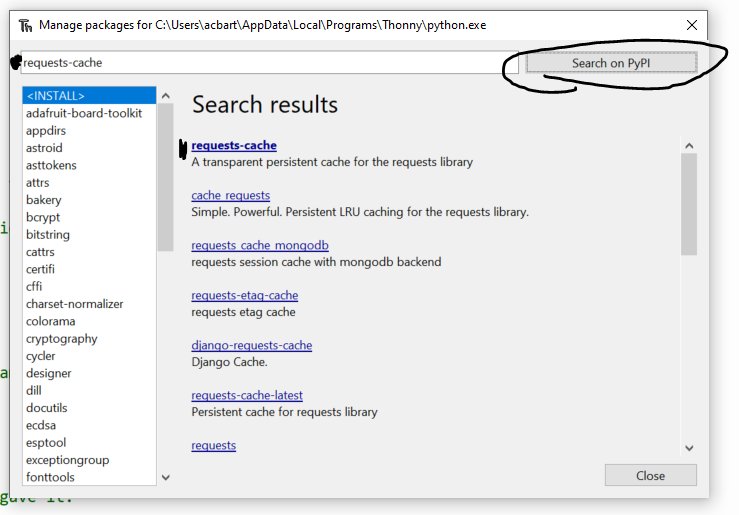
# 1) Setup

In order to complete today’s activity, you must install certain packages and organize your filesystem so that we can import some custom modules.

All members of the group need to follow these steps, so that their computer is setup correctly.

1) First we need to install the new package:

1. Open up Thonny. In the menu at the top of the window (top of screen on Mac), click “Tools” and then “Manage Packages”.



1. In the new window, type “requests-cache” without quotes
2. Click “Search on PyPi” button in the top-right corner
3. Click the first bolded result, which is the “requests-cache” library.
4. Click the Install button.
5. Make sure the Install button changes to a disabled “Upgrade” button. Sometimes Thonny will act like the library installs but actually needs to be run twice.
6. If there are issues with the installation, ask for help.
7. Each member of the group should [take a screenshot](https://www.take-a-screenshot.org/) of the Manage Packages window once the library has been successfully installed, and paste it into the table cells below. Keep the images small so that all three fit comfortably in the same row.

|  |  |  |
| --- | --- | --- |

2) Next we need to organize your file system so that you will know where everything is. If you know how to navigate files and folders on your computer, these next steps may be easy. However, if you are not used to them, you should read the next steps carefully.

1. First, open up the [File Explorer](https://support.microsoft.com/en-us/windows/find-and-open-file-explorer-ef370130-1cca-9dc5-e0df-2f7416fe1cb1) (windows) or Finder (mac,  on the dock)
2. Navigate to your Documents folder, or whatever folder you want to use to store important course files. Do NOT store important files in your Downloads folder.
3. You will need to create at least one folder for your course work files (e.g., cisc108). We really recommend a few levels of nesting so that you can keep all your coursework separate between classes and projects.  
   For example, this is how the course designer organized folders for this class and assignment in particular.  
     
   The *relative* path to this assignment’s folder from our system Documents folder is:  
   school/cisc108/lecture\_10\_requests  
     
   But the full *absolute* path to the folder is:  
   C:/Users/acbart/Documents/school/cisc108/lecture\_10\_requests
4. To finish up this step, everyone will need to get the full absolute path to the folder where they plan to store files for this assignment:
   1. On Windows:
      1. Hold shift key down
      2. Right click the folder
      3. Click Copy as Path
      4. Paste the filename in the table below without quotation marks
   2. On Mac:
      1. "Right-click" the folder (either two-finger tap or hold down the control key while clicking)
      2. Hold ⌥ key down (option/alt)
      3. Click Copy "item" as Pathname
      4. Paste the filename in the table below

|  |
| --- |
|  |
|  |

3) Back in Thonny, we have a little more setup to do.

1. In the menu at the top of the window (top of screen on Mac), click “View” and then click “Files”.
2. The Files tab should now appear on the left side of the application window.
3. Make sure you can also see the “Shell” and “Variables” panels (you can use the View menu to make sure they are visible). It is up to you if you want other panels visible
4. In the menu at the top, click “File” and then click “New”. A new blank file appears.
5. In the menu at the top, click “File” and then click “Save As”
6. In the window that appears, you must successfully navigate to the new folder you previously created to hold your course work files.
7. When you are in the folder, enter the filename as “basic\_requests.py” (without quotes) and then click “Save”.
8. Click the “Run” button to run the blank file. The “Files” panel on the left should change to only show the “basic\_requests.py” file (since that folder is otherwise empty).
9. Take a screenshot of the entire Thonny window demonstrating that you have successfully set up the project, and paste the screenshot below. Remember the process we followed for creating a new file so that you can make more new files in the future!

|  |  |  |
| --- | --- | --- |

# 2) Basic Requests

Now that we have successfully set up your system, we can start working with basic requests.

Copy the following code into your “basic\_requests.py” file to use as a base.

| # One-time import of requests library  import requests  # Get data from a URL  response = requests.get("")  # Print out the content of the response  print(response.text) |
| --- |

1) Run the code we just gave you, and read the lengthy error message that occurs. You need to learn how to read complicated error messages like this one, so identify the following:

1. The last line of the error message tells you what kind of error occurred. Copy and paste only the last line of the error message.

|  |
| --- |

1. The lines above the last line are the Traceback, or a record of what functions were being called when the error occurred. Each function call adds two lines to the traceback: one line indicating what file and line the function call occurred in, and then the actual line of code with the function call. The records are in the order that the functions were called in.  
     
   The majority of the records in the traceback are referencing files that you did not write, because most of the function calls happened in the Requests module that we imported. However, the error is still in our code, rather than the Requests module. Read over the Traceback and look for the record that references YOUR file, and then copy/paste the two lines of that record below (make sure you are not tricked by long lines spilling over onto multiple lines; one record may appear to take up more than two lines).

|  |
| --- |

1. We seem to have made a mistake in our example. What was the mistake that was made that caused the error?

|  |
| --- |

2) Copy the following URL and paste it between the double quotes in your *basic\_requests.py* to replace the empty string.

https://httpbin.org/base64/SGVsbG8gV29ybGQh

Run the code and copy the output below:

|  |
| --- |

3) Now let’s try another URL. Copy the second and third lines of actual code that we gave you before (where we made the request and printed out the response). Replace the current URL with the following URL:

https://httpbin.org/user-agent

What is the output of requesting this URL?

|  |
| --- |

4) One more fun URL! Copy the request/print lines again, and then use the following URL:

https://pastebin.com/raw/V7tWn1Rj

What is the output of requesting this URL?

|  |
| --- |

5) Now we’re going to try requesting some real time web data. Copy the request/print lines again, and then use this URL:

https://forecast.weather.gov/MapClick.php?lat=39.6086&lon=-75.8301&FcstType=json

Copy the entire output to the box below. There will be a lot of it! If you read the output carefully, you can see a lot of useful data. We just need to tell Python how to make sense of the structure.

|  |
| --- |

# 3) Weather Requests

Web requests often respond with data in a format called “JSON” (short for "JavaScript Object Notation"). This format is a carefully crafted (usually long) string which can be "easily" turned into the types of data that we’ve learned this semester: integers, strings, floats, booleans, lists, and types made with dataclasses. However, we want to skip over the details of how to convert this data for now, so we’ve prepared a library that will do a lot of the annoying little steps and quickly convert web-based data into a format you already know how to work with.

1) You will need to download our library and carefully place it in the same folder that you created earlier. If the library file is not in the right folder, then nothing will work correctly. Every time you want to use the library from another file, the two files must be in the same folder.

1. Open the latest version of the Bakery Weather library here: <https://gist.githubusercontent.com/acbart/e9f7510631bc56d3f0378de9b2b8f3ce/raw/bakery_weather.py>
2. "Right click" and save the page to the coursework directory you created previously.
3. In Thonny, you should see the file in the Files panel of Thonny. Make sure that the name of the file is exactly “bakery\_weather.py” with no additional text (you may need to right-click and rename the file to remove any stray “.txt” that may be added).  
     
   If the file does not appear correctly, then delete the file and follow these steps insteads:
   1. Create a new file in Thonny
   2. Select all of the code from the linked version of the library above and paste it into the new file.
   3. Save the file as “bakery\_weather.py” (without quotes) in the coursework directory.
4. When you are done, take a screenshot of the Files panel on each computer to prove that everything is organized correctly.

|  |  |  |
| --- | --- | --- |

2) Create a new file in Thonny named “weather.py” and place it in the same coursework directory that you have placed your other files. Copy and paste the following code:

| from bakery\_weather import get\_weather  weather = get\_weather(39.6782, -75.7616)  print(weather.current.temperature) |
| --- |

This code uses the get\_weather function (which consumes two floats representing the latitude and longitude of a position in the United States) to get a WeatherReport object, and then accesses the data inside of that object to print out the current temperature.

Run the code and copy the output below.  
If there is an error, you will need to revisit the steps above to make sure the library was downloaded correctly. Seek help from course staff if you cannot resolve the issue!

|  |
| --- |

3) So how did we know that we could use the get\_weather function, and what kind of data would be returned? The answer is **documentation**. The bakery\_weather.py library file provides some documentation on the function and dataclasses inside as *triple-quoted docstrings*. You will need to read this documentation to know how to use future libraries. Sometimes the documentation is outside of the code, and sometimes the documentation is inside of the code. To answer the following questions, you will need to refer to the documentation inside of the code.

1. Look carefully through the file for the definition of the get\_weather function. Then, fill out the following table with the relevant information:

| **First parameter name** | latitude |
| --- | --- |
| **First parameter type** | float |
| **Second parameter name** |  |
| **Second parameter type** |  |
| **Return type** |  |
| **Docstring** |  |

1. Look carefully through the file for the WeatherReport dataclass. Then, fill out the following table with the relevant information:

| **First attribute name** | location |
| --- | --- |
| **First attribute type** |  |
| **First attribute description** | Metadata about the location this forecast comes from. |
| **Second attribute name** |  |
| **Second attribute type** |  |
| **Second attribute description** |  |
| **Third attribute name** |  |
| **Third attribute type** | list[Forecast] |
| **Third attribute description** |  |
| **Overall docstring** | Main actual weather report class that contains the location of the result, the current weather, and the forecasted weather. |

1. We need to also get the documentation for two other classes (there’s actually a fourth class, but we’ll ignore that one for now). The next important class is WeatherData. We’ll start writing the table more concisely so that we can save ourselves some space.

| **WeatherData**  The current weather information, which is more extensive than what can be forecasted. | | |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| temperature | int | The current temperature (in Fahrenheit). |
| dewpoint |  | The current dewpoint temperature (in Fahrenheit). |
| humidity |  | The current relative humidity (as the whole number part of a percentage). |
| wind\_speed |  |  |
| wind\_direction |  |  |
| description |  | A human-readable description of the current weather. |
| url | str |  |
| visibility |  | How far you can see (in miles). |
| wind\_chill |  |  |
| pressure |  | The barometric pressure (in inches). |

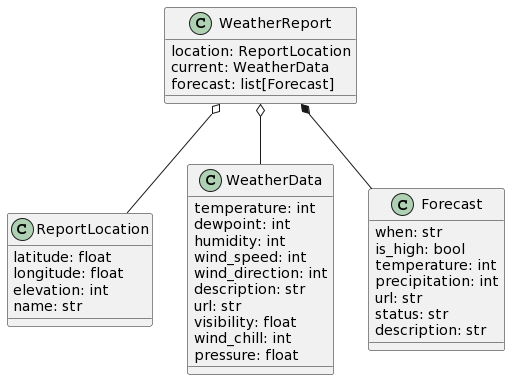
1. The last class that needs documentation is the Forecast class. Fill out this one on your own!

| **Forecast**  The weather predictions for a given time period. | | |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. Before we wrap up looking at the documentation, we need to make sure we understand the relationship between these classes. The main one returned from our get\_weather function is WeatherReport. That class uses three other classes. However, one class is different from the others, in terms of how it is used in the structure of WeatherReport. Which class is different and how is it different?

|  |
| --- |

You may find the following class diagram helpful:



4) The time has come to actually start analyzing the web data that we have been accessing! First, we will write a function named get\_celsius that consumes a latitude and longitude (both floats), gets the current weather at that location, and then returns the temperature in Celsius (instead of Fahrenheit) as a float. Note that we will not require you to unit test this function, because of the dynamic nature of web data. Instead, you will just call the function multiple times for different locations and print the results directly.

Write code in the weather.py file, not the bakery\_weather.py file. Do not edit the library!

The formula for converting from Fahrenheit to Celsius is:

Copy/paste your completed code for the function in the box below:

| def get\_celsius(latitude: float, longitude: float) -> float: |
| --- |

Then, use the function to fill out the temperature in Celsius for the following locations:

| **Latitude, Longitude** | **Location Name** | **Output** |
| --- | --- | --- |
| 39.6782, -75.7616 | Newark, DE |  |
| 21.4395, -157.9436 | Honolulu, HI |  |
| 64.499, -165.3988 | Nome, AK |  |

5) Now let’s try working with some of the more nested data. Define a function named get\_average\_temp that consumes a latitude and longitude (both floats), gets the forecast using get\_weather, and then produces a float representing the average forecasted temperature (in degrees Fahrenheit) as a float. Again, you do not need to unit test this function.

Copy/paste your completed code for the function in the box below:

| def get\_average\_temp(latitude: float, longitude: float) -> float: |
| --- |

Then, use this function to fill out the average forecasted temperature for the following locations:

| **Latitude, Longitude** | **Location Name** | **Output** |
| --- | --- | --- |
| 39.6782, -75.7616 | Newark, DE |  |
| 21.4395, -157.9436 | Honolulu, HI |  |
| 64.499, -165.3988 | Nome, AK |  |

6) Next, define a function named count\_sunny that consumes a latitude and longitude (both floats), gets the forecast, and then produces an integer representing the number of forecasted reports where the status includes the text “Sunny”. Since the status can be both exactly “Sunny” but also things like “Partly Sunny” and “Mostly Sunny”, you will need to check if the string “Sunny” is in the status field’s value.

Copy/paste your completed code for the function in the box below:

| def count\_sunny(latitude: float, longitude: float) -> int: |
| --- |

Then, use this function to fill out the number of sunny days for the following locations:

| **Latitude, Longitude** | **Location Name** | **Output** |
| --- | --- | --- |
| 39.6782, -75.7616 | Newark, DE |  |
| 21.4395, -157.9436 | Honolulu, HI |  |
| 64.499, -165.3988 | Nome, AK |  |

7) Now for a tricky one! Define a function named coldest\_description that consumes a latitude and longitude (both floats), gets the forecast, and then produces a string representing the description of the coldest (lowest temperature) forecast entry based on the *high* values. To determine whether a Forecast is a high or a low, use the is\_high boolean field.

Copy/paste your completed code for the function in the box below:

| def coldest\_description(latitude: float, longitude: float) -> str: |
| --- |

Then, use this function to fill out the description of the coldest day for the following locations:

| **Latitude, Longitude** | **Location Name** | **Output** |
| --- | --- | --- |
| 39.6782, -75.7616 | Newark, DE |  |
| 21.4395, -157.9436 | Honolulu, HI |  |
| 64.499, -165.3988 | Nome, AK |  |

# 4) Extra Practice

If you manage to reach this section, then the following problems are extra credit. Instead of using the Bakery Weather library, they instead use the Bakery Reddit library. You will need to set up that library just as you did the Weather library, and read over its documentation in the same way.

*Note: Although we have attempted to filter out NSFW results, the internet is a wild place and you never know what you will find. Please be mature and avoid intentionally searching for inappropriate content during class. If you find anything rude or inappropriate, just ignore it and move on. You can delete the “bakery\_cache.sqlite” file if you want to get rid of previously downloaded data and get new fresh content instead.*

Begin by downloading the library: <https://gist.githubusercontent.com/acbart/d257e8ac419fe6c28425b80cdc19e620/raw/bakery_reddit.py>

1) Define a function average\_score that consumes the name of a subreddit and a sorting style (both strings), gets the latest posts, and then produces a float representing the average score of the posts.

Copy/paste your completed code for the function in the box below:

| def average\_score(subreddit: str, sorting: str) -> float: |
| --- |

Then, use this function to fill out the average score for these different subreddits.

| **Subreddit, Sorting** | **Output** |
| --- | --- |
| “programming”, “top” |  |
| “python”, “top” |  |
| “learnprogramming”, “top” |  |
| “compsci”, “top” |  |

2) Define a function count\_python that consumes the name of a subreddit and a sorting style (both strings), gets the latest posts, and then produces an integer representing the number of times that the string “python” appears in the content of each Post that is NOT a URL (you can use the is\_url field). Use the [.count(term)](https://www.w3schools.com/python/ref_string_count.asp) and [.lower()](https://www.w3schools.com/python/ref_string_lower.asp) methods of strings to ignore capitalization.

Copy/paste your completed code for the function in the box below:

| def count\_python(subreddit: str, sorting: str) -> int: |
| --- |

Then, use this function to fill out the counts for these different subreddits.

| **Subreddit, Sorting** | **Output** |
| --- | --- |
| “programming”, “top” |  |
| “python”, “top” |  |
| “learnprogramming”, “top” |  |
| “compsci”, “top” |  |

3) Define a function of your own choosing that consumes the name of a subreddit and a sorting style (both strings), gets the latest posts, and then produces some data that is interesting. You are free to decide what you will analyze, and what subreddits and sorting styles you will use.

Copy/paste your completed code for the function in the box below:

| def \_\_\_(subreddit: str, sorting: str) -> \_\_\_: |
| --- |

Then, use this function to fill out the counts for different subreddits and sorting styles.

| **Subreddit, Sorting** | **Output** |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

# 5) Reflect and Review

Discuss among yourselves: what did you learn from this activity? What was surprising or interesting? If you didn’t learn anything, what do you think we were trying to teach you? How could this activity be improved?

|  |
| --- |
|  |
|  |

# Final Submission

When your team is happy with your answers for all the questions, download this file as a Word Document (docx) and upload the file to the appropriate assignment on Canvas.

Only one member of your team needs to submit.