# CS150 - Assignment 9
# Weather or not...
# Due: Wednesday Nov. 14, at the beginning of class


An important component of many scientific applications is data collection and data analysis. For this lab, we'll be looking at an example data collection application that collects weather data from the web and aggregates it into a data file. In addition, we'll also make a nice script that you takes a zip code as a command-line parameter and will give you the current temperature.

**A few disclaimers:**

1. As I mentioned in the class video lecture, when dealing with a program like this that contacts an external server, *you need to be respectful of that external resource*. For testing purposes, I have put up a version of the web page you will be extracting the temperature from on our department web server. You should use this test web page until you have your program working. Even when you have your program working and change over to the external web address, please avoid making too many repeated calls.

2. If you wanted to write a program like we're writing below for a real application, you would not "scrape" the web page for the content you needed. Instead, many companies offer what is called "API" access (application programming interface) to their data. This would be a module you would download and use and it would give you the data. We're only doing it through scraping for educational purposes. If you want to actually do it the right way at some point, please come talk to me.

# 1 Part 1: Getting the weather

For the first part of this lab, you are to write a program that reads the current weather from the web for a zip code entered by the user. I've broken the description of this program into two parts: the specification of what is required, and my suggestion about how to proceed on the implementation. Make sure to read both sections before starting!

## 1.1 Specifications

Write a program called `weather_reader.py` that has the following characteristics:

- Importing the module does nothing besides import any functions/variables that are defined in your module.
- Your program should be able to be run from the command-line and take a single argument, which is the zip code:
  - If your program is run with too few or two many arguments, it should print out the usage:

    ```
    dkauchak$ python weather_reader.py
    weather_reader.py <zip_code>
    ```

  - If it is run with the correct number of arguments (one) you should treat it as a zip code (you can assume it's correct) and the program should print out the current temperature at tht zip code

    ```
    dkauchak$ python weather_reader.py 05753
    59
    ```

- Your module *must* contain a function called `get_temperature` that takes a zip code as a parameter and returns the temperature at that zip code. For now, this version will be ignoring the zip code and extracting the temperature from our test web page.

## 1.2 Implementation

At:

`http://www.cs.middlebury.edu/~dkauchak/classes/cs150/assignments/assign9/weather_example.html`

I have included a snapshot of a query from `weather.com`. *For now, your program should only use this page.* If you open up this web page you'll see it's the weather for Middlebury. To get our weather we're going to be extracting the weather from the html of this page.

You may implement this module however you like as long as it meets the specifications above, however, here is one suggested approach to implementing it:

1. Write the part of the program that checks to see if this program is being run vs. imported, checks the number of program parameters and prints the usage accordingly.

2. Write some code that opens the web page above and reads through it a line at a time (when I say "code" I mean some statements that may be stand-alone or may be in one or more functions).

3. Once you have this working, you now need to actually find the temperature in the web page. Go to the web page above and view the source. You'll see there is a surprising amount of text to generate this one page (13,908 lines to be exact).

   The fortunate thing is that we just need one small piece of information, the current temperature. In the source file search for the following phrase (using "find" under the edit menu):

   ```
   "temperature-fahrenheit">
   ```

   You'll notice that immediately following this string, is the temperature (it actually occurs twice in the html file).

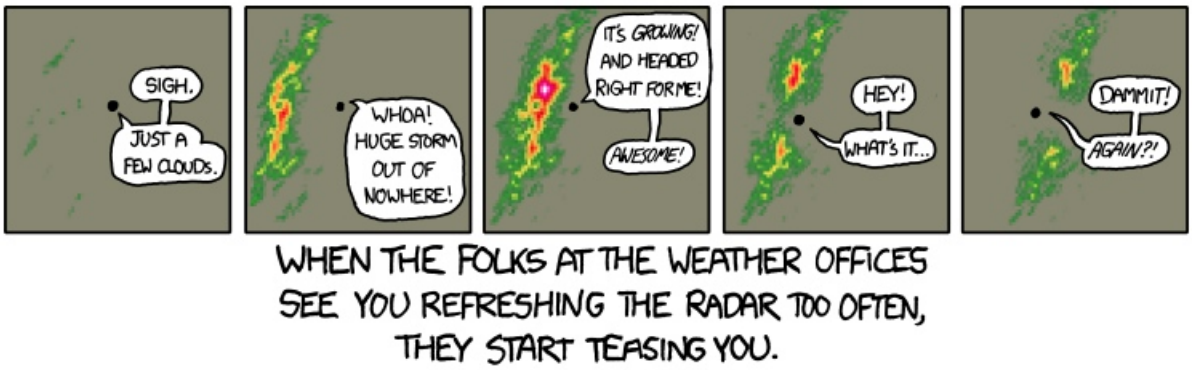
   Modify your code to find the line that contains this string. You can print it out just to make sure you've got the right line.

4. Once you have the right line with the temperature in it, you'll need to extract the actual temperature number. Put this all together to write the `get_temperature` function. Recall that it will take a zip code as a parameter. For now it will just ignore that parameter and always get the temperature data from the above web page, but go ahead an put the parameter in anyway.

5. Finish up your program so that when you run it with the correct number of arguments it prints out the temperature.

You should now be able to run your program from the command-line with a zip code and it will give you a temperature (pretty cool!). Right now, it should always give you 34, however, it will just be a small change to have it do the real thing. We'll get to that soon...

## 2 Part 2: Aggregating the weather




We now have a program that we can run and it gives us the temperature ***and*** we have a module that we could import and call the `get_temperature` function to get the current temperature with a zip code. For the second part of this lab, we're going to write *another* program (i.e. in a different .py file) that can be run regularly over time to build up a file with aggregated temperature data over time.

Your program will be run with two command-line arguments, the name of a file and a zip code. The file will contain multiple entries collected over time. Each line in the file will consist of a date, an hour of the day and the temperature at that hour. For example, here is a short snippet of an example file:

```
11-8-2012       14      65
11-8-2012       15      66
11-8-2012       16      70
11-8-2012       17      68
11-8-2012       18      59
```

Each time you run the program it will **add at most one line** to this file. So the file above would have been generated with at least five calls to the program (over 5 different hours).

We're setting the problem up this way since it is generally straightforward to get a program to run at some fixed interval. You won't be doing that for this lab, but I'm happy to talk to you offline about how that would work.

As with the first part, I've broken the description of this program into two parts, the specification and the implementation.

### 2.1 Specifications

Write a program called `weather_aggregator.py` that has the following characteristics:

- Importing the module does nothing besides import any functions/variables that are defined in your module.
- Your program should be able to be run from the command-line and take two arguments, the first a filename and the second a zip code
- If the program is run with more or less command-line arguments it should print out the usage

```
dkauchak$ python weather_aggregator.py
weather_aggregator.py <file> <zip_code>
```

- If the program is run with two arguments:
  - The program should first check to make sure that there isn't already an entry in the file for the current date and hour. If there is, the program should do nothing. This means that running the program repeatedly in a row will not alter the file after the first time when the current temperature is added for the current hour.
  - If there is not an entry in the file for the current date and hour, the program should use the `weather_reader` module to get the current temperature and *add* an entry to the file at the end with the appropriate formatting (i.e. date, then hour, then temperature all separated by tabs).

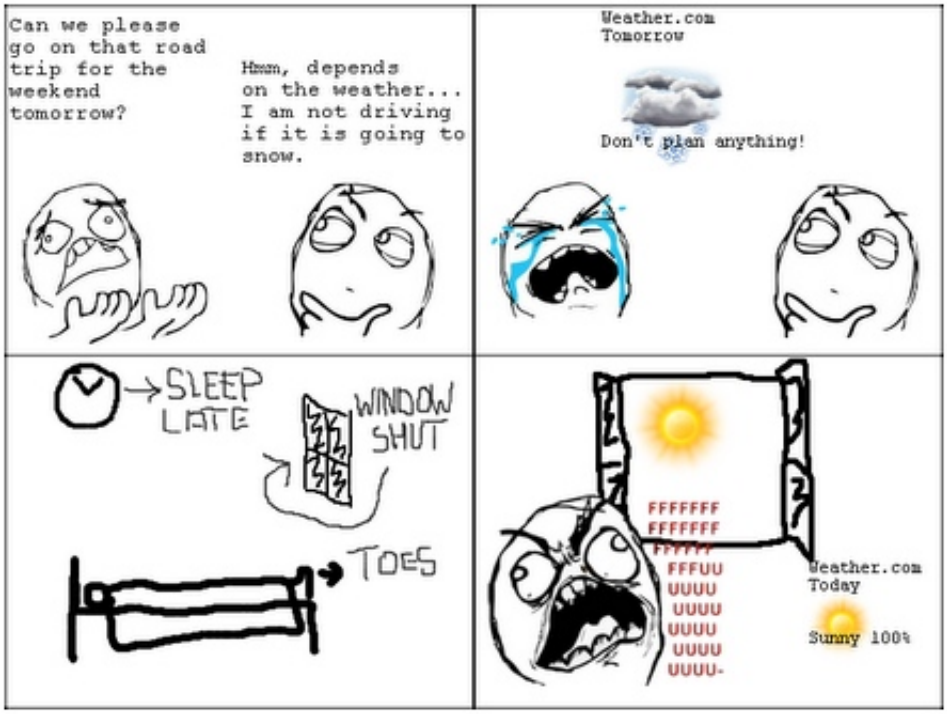
### 2.2 Implementation

Here is one approach to implementing this program:

1. Write the part of the program that checks to see if this program is being run vs. imported, checks the number of program parameters and prints the usage accordingly.
2. Write some code to check whether the file has an entry for a given date and time. For testing purposes, it will likely be useful to create a version of the aggregate file manually (e.g. in Wing or another text editor).
3. Write some code that gets the current date and hour and checks to see if it's in the file. Again, just modify the file by hand to test this.
4. Finally, put this together so that you check to see if the current date is in the file already, if it's not, use your `weather_reader` module to get the temperature and *append* it on to the end of the file. When writing this file, you may either rewrite the entire thing from scratch each time (in that case you'd open the file with "w") or you can just append the one new entry (in that case you'd open the file with "a"). *Don't forget to close your files after you write out the data.*
5. Add any finishing touches to the program to make sure it runs appropriately. When you run it, you won't see any output, but the file passed in may change.

**NOTE:** If the file doesn't exist at all and you call your program, you may get an error. To get around this, just create a blank text file with nothing in it before you run your program for the first time.

## 3 The real deal




http://blog.anarchius.org/2011/01/weathercom-rage-comic.html

So far, all of your testing should have been done on the copy on the computer science web server using the url above, always giving you the same temperature. When you're confident that you have everything working you can go back and change your `weather_reader` module to use the real web page.

For a given zip code, the url should consist of

http://www.weather.com/weather/right-now/

with the zip code appended on to the end. For example, if we were to query Middlebury's weather, the url would be:

http://www.weather.com/weather/right-now/05753

Change your `get_temperature` function in the `weather_reader` module to generate an appropriate url based on the zip code passed in and then use this url to get the temperature.

You should now be able to query the current weather based on the zip code entered:

```
dkauchak$ python weather_reader.py 05753
67
dkauchak$ python weather_reader.py 92037
56
dkauchak$ python weather_reader.py 84109
24
```

Again, please try not to run this program too many times, but do play with it some. You should be able to run your `weather_reader.py` program with a zip code and it will give you the current temperature and your `weather_aggregator.py` should now aggregate the real values.

## 4 Extra credit

You may earn up to 2 points of extra credit on this assignment. Below are some ideas, but you may incorporate your own if you'd like. Make sure to document your extra credit additions in comments at the top of the file.

- Check to make sure that the user enters a valid zip code (i.e. 5 digits)
- Modify your `weather_aggregator.py` program so that it will run the first time even if the file doesn't already exist.
- Also include the zip code in the aggregated file and add data to the file based on whether an entry for that date, time and zip code do not exist in the file.

## 5 When you're done

Make sure that your program is properly commented:

- You should have a docstring comment for each module at the top of the file
- You should have comments after the module doc string stating your name, course (including section number), assignment number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good ***style***

### Submission procedure

For this assignment, you will have two .py files to submit. To submit multiple files, create a directory with your name followed by the lab number. For example, my folder would be called `davidkauchak9` (*but use your name!*).

Put your two .py files inside this directory and the create a .zip file from this directory. On the Macs, right-click on the directory and select "Compress...". If you're working on Windows, right-click on the file and select "Send to" then select "Compressed (zipped) Folder" (or if you don't have this option, use Winzip (`http://www.winzip.com/`). You will then see a file with a .zip extension be created.

Submit this .zip file digitally at the usual location.

---

**Grading**

| | | points |
|---|---|---|
| `weather_reader.py` | | |
| | run vs. import | 1 |
| | prints usage with incorrect number of arguments | 2 |
| | runs correctly with zip code entered | 2 |
| | `get_temperature` | 5 |
| `weather_aggregatore.py` | | |
| | run vs. import | 1 |
| | prints usage with incorrect number of arguments | 2 |
| | data and hour formatted correctly in file | 1 |
| | appends temp to end of file | 3 |
| | doesn't add repeated data | 3 |
| Comments, style | | 5 |
| extra credit | | 2 |
| total | | 25 (+2) |