The following two problems have corresponding templates ready for you to get started on Github. Do not forget to adjust the README to indicate you have completed the assignment before your final commit! There are no written problems this week, so you won't need to scan anything to submit.

Get Assignment link: https://classroom.github.com/a/9ujox579

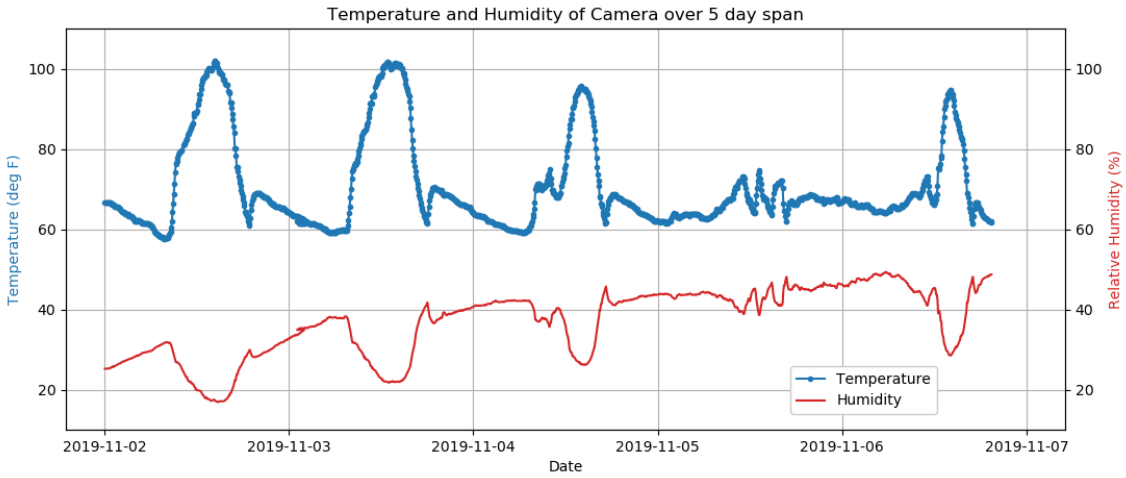
1. In some cases, we might not even need Python to generate or compute the numbers that we want to display. Sometimes, we just need Python to help us process and then visualize numbers or values that we have gotten from other sources. The file `Conditions.csv` is a comma-separated list of temperature and humidity readings from inside the enclosure of my research camera system. Each line of the file contains a date and time stamp, the temperature in degrees Fahrenheit, and the relative percent humidity. Your task in this problem is to read in the data, ensure that it is in a form that Python can understand, and then visualize the final temperature and humidity over the given 5 day span.

   (a) Your initial goal will be to read in all of the data and construct lists of the date-time, temperature, and humidity data. There are a few ways to do this, but one way would involve opening the file and then looping through the file line by line. For each line, strip off any leading spaces or trailing new-line characters, and then split it at the commas. That will get you the three necessary parts that you'd want to append to your lists.

   There are some important filtering considerations to keep in mind! Occasionally, when asked for a temperature and humidity, the sensor will not reply quick enough. In this case those entries are left empty. You do *not* want to append the data (date, temperature, and humidity) if *any* of the entries are empty! Additionally, occasionally the sensor will report a highly erroneous pair of temperature and humidity values, typically with a relative percent humidity which is well above 100 (which is impossible). So you should also not append data in cases when the humidity at that time is reported to be greater than 100.

   Finally, a note on variable types. Temperature and humidity should be floats. The time-stamps will naturally be a string, but Matplotlib will not know how to treat them in this case. Instead, you will want to import the datetime module, and convert the time-stamps to a date-time object. Matplotlib can treat objects of this type correctly, and it will make it easy to get the correct sorts of units on your x-axis when it comes to plotting. In particular, the `strptime()` function is what you will need to use to convert the string to the proper datetime object. This function requires you giving it a template for the form that the date is written. It will be useful to consult here for some info on the various template formats so you can construct a proper template. I will include some examples in the template code.

(b) Once you have the data in properly formatted lists, you can proceed to create the plot. Since we want to plot both temperature and humidity together, we will use a shared x-axis (the datetime) with one y-axis belonging to temperature and the other belonging to humidity. You'll most likely want to use `.twinx()` to create the second, twinned axis. Then you can add the individual plots and plot annotations and labels. Your goal is to duplicate the below plot as closely as possible.



You should use `<figure obj>.savefig(filename)` to save your figure *before* you use `plt.show()` within your script. Call the file `Conditions.png`.

2. We have used the `arcade` library throughout the semester, but we have not really scratched the surface as to its true powers and flexibility. For that, we needed knowledge of classes, which we know have! So in this problem we will take advantage of the the built in classes of `arcade` to easily add mouse and keyboard control to a simple scene.

   (a) Create a new class called `Prob2` and have it inherit from the `arcade.Window` class. This is the type of object you have been creating earlier in the semester every time you would type: `arcade.open_window()`. We will be taking advantage of many more aspects of this class type in a moment, but for the time being we want to recreate the basics.

   Define a new `__init__` function which takes a width, height, and window title as input (sound familiar?). Go ahead and use the inherited `arcade.Window.__init__` method to set up the basic window. Afterwards, declare a background color for the window as well, just like we have done in the past. Your choice on color!

   To make sure everything is working initially, add a `on_draw` method. This is the method that is called whenever the window class wants to draw something on the window. Inside, include the basic drawing commands to start the render and then draw a simple filled circle somewhere on the screen. We will be modifying this circle much more in coming parts, but we just want to make sure the basics are setup here. So choose any position, size, and color.

   I have already written a `main` function for you further down in the code which will create the necessary object using your new class and then actually run it. So if you run your code now, you should get a window of the correct background color and with a circle drawn somewhere on it. Make sure that this much is working before going on to the next parts!

   (b) We want to add the functionality for the circle to follow your mouse cursor around the screen. To do so, we will override the class method `on_mouse_motion` with a few simple lines of code. Realize that at the moment, the function where you *draw* the circle (`on_draw`) and the function where you would change or update that drawing (`on_mouse_motion`) are different functions with their own scopes! But `self` is passed in as an argument to both (since they are both methods of the class), so the easiest way to have a variable shared between them would be if it was added to the `self` "bucket". So in your `__init__` function, add two data attributes called `self.circle_x` and `self.circle_y`. Then update your code in `on_draw` where you draw the circle to use these attributes. And now in your code in `on_mouse_motion` update these attributes to be whatever the current x and y position of the mouse are. Now if you run the program and move your mouse about the window, the circle should follow it!

(c) Let us also the add ability so that when the user left-clicks, the circle will cycle between 3 different colors. So one click advances it to the second color, another click advances it to a third color, and then another click cycles it back around to the first color. The plan here is to override the method `on_mouse_pressed` to update the color whenever the left mouse button is pressed. Doing so will require adding some more data attributes to your class in `__init__`, (at the very least something to keep track of the current color, but probably a list of 3 colors as well. And maybe a current color index). You can choose any colors you want, but make sure each time you click it advances to the next color.

(d) Finally, let us add an actual nice way to exit the window. Add a `on_key_pressed` method to check for if the "q" key is pressed. If it is, then you can always close a window by running the `arcade.close_window()` function. Implement that here so that when "q" is pressed, the window closes immediately.

Like problem 1, there is no autotesting for this problem so make sure you test it yourself and ensure it is behaving properly and not causing any errors!