Lab 4a: Preparing data for the hurricane tracking lab

Due at 7 pm on Monday 23 September:

1) A spreadsheet with summaries of the advisory information for your assigned hurricane
2) Three weeks of data sources in your portfolio spreadsheet

None of this needs to be handed in, but you need to be ready to work on the lab in class on Monday.

1. Setting up
    a. Create a folder on your external drive called **Lab4**. Create a sub-folder called **tracks**.

2. Hurricane advisories
    a. On the next page will be a set of directions for how Joe has made it much easier for you to get the advisories for your hurricane using Python.
3. Data spreadsheet for your portfolio
    a. Finally, you need to fill in your data spreadsheet (see the Course Overview section on Blackboard) for the data you prepped today. While you're at it, update last week's data as well if you haven't done so yet. Save the file to your drive with a name and location that you can remember. Don't worry if you can't fill in all the parts of the data for today. You should know all the parts for the data from last week.

Loops

September 1, 2016

A note on future exercises

In the first exercise, I tried to speak intuitively rather than use specific Python vocabulary words. However, some knowledge of the standard vocabulary can be immensely helpful when looking for help online, so now I will begin to transition from more intuitive definitions to more typical Python vocabulary. Specific vocabulary words will be **bolded**, and might be good additions to the glossary in your portfolio.

1 Patterns

In this lab you will be working with hurricane data from NOAA. NOAA archives all the advisories they issue for hurricanes and tropical storms.

1. Go to http://www.nhc.noaa.gov/help/tcm.shtml?ALL and go over the description of their advisories. You will use the information in the advisories to track the position of the hurricane, its size, and the forecasts over the course of the storms life.

2. See the annotated advisory provided to see what information you will be using and where in the advisory it comes from (Fig. 1).

3. Now, go to http://www.nhc.noaa.gov/archive/2016/ and find the advisory archive for the hurricane you will be working with.

4. Look at a few of the advisories and try to locate the information we're interested in.

   Now you could go through every advisory, copying down the information you need, but there's a better way! Each advisory is structured the same way and contains the same information. All the advisories together can be thought of as forming a pattern, repeating again and again with slight variations. You can teach Python this pattern by creating a template, and like playing a game of Mad Libs in reverse (Fig. 2), use the template to pick out the hurricane's latitude like you could the first custom adjective used in a completed Mad Lib. Although the construction of these

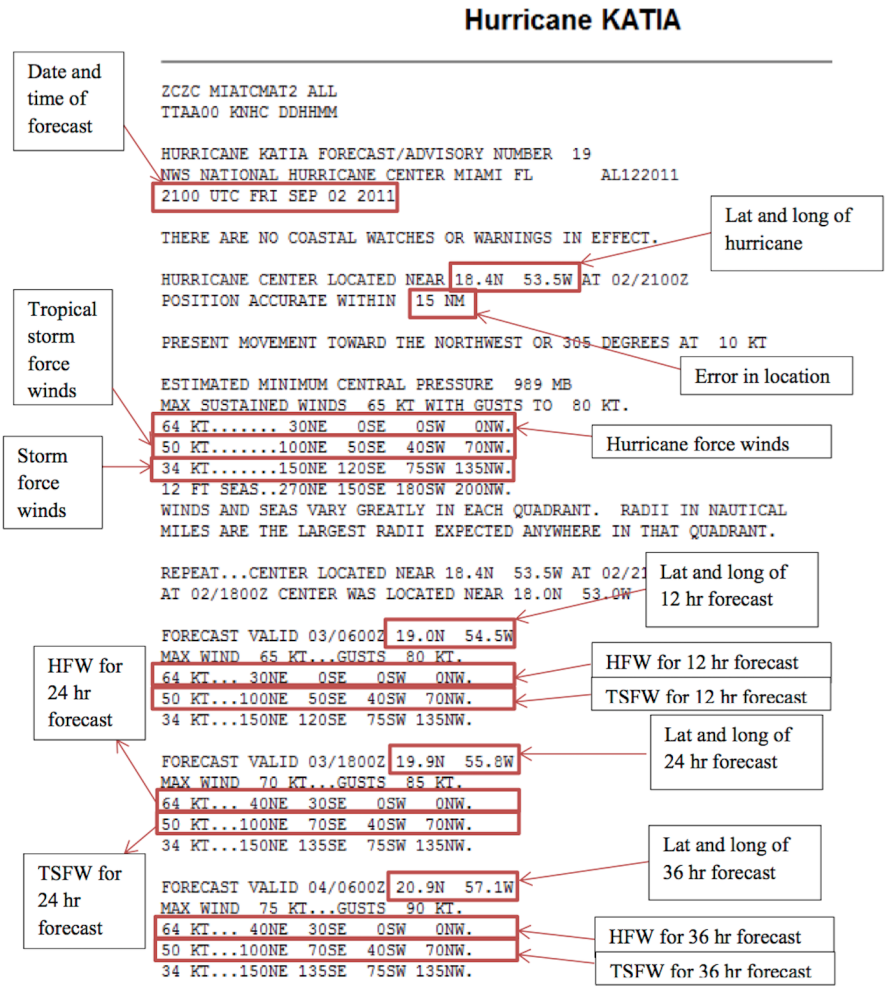

Figure 1: Example hurricane advisory

templates is slightly outside of the scope of this course[1], start trying to recognize times you work with any sort of text in a repetitive manner, like finding all the phone numbers or emails in a document or changing the way a date or title is formatted in a set of documents. These could be situations where a little script could save a lot of time.

2 The script

1. Download and extract the zip file containing this weeks exercise from blackboard.

2. Open up `exercise_2.py` with IDLE.

   Hopefully this is starting to look a little familiar, we'll go through the scrip line by line and introduce three new topics: comments, file paths, and loops.

3. `# This script...` The `#` symbol tells Python to ignore everything until the next line. This allows you to write **comments** in your scripts for any people who might be reading it.

4. `import forecast_parser as fp` Don't worry about this line for now, it's just telling Python where to find some of the tools we'll be using in this script.

5. `ADVISORY_ARCHIVE_URL = "" # The...` See that `=` sign? If you'll recall from last week, this means we're setting a label.

   (a) In the context of Python, labels are referred to as **variables** since what exactly they refer to can *vary*. Think back to algebra, the concept of a variable in Python is essentially the same as the variables you worked with in math, although in Python, variables can refer to more than just numbers.

   (b) The label name is `ADVISORY_ARCHIVE_URL`, but what exactly are we labeling? Single quotes (`'`) and double quotes (`"`) are used to delineate **strings**, blocks of text. This distinguishes it from text that Python should try to interpret, like labels and tools and instructions.

   (c) There's nothing in between the two quotes, so right now the variable `ADVISORY_ARCHIVE_URL` is referring to an "empty" string, no text at all. Having a variable refer to something is called **assignment**. In this instruction, a variable is **assigned** the value of an empty string.

   (d) The `#` designates a comment, so Python knows to ignore the rest of the line.

[1] The way that the templates were defined for this script is called "regular expressions", or regex. They can be a bit obtuse but with a reference and some trail and error, you can get a lot done. If you're interested in learning how to write them, here are some fun ways to get started: regex crosswords, interactive tutorial

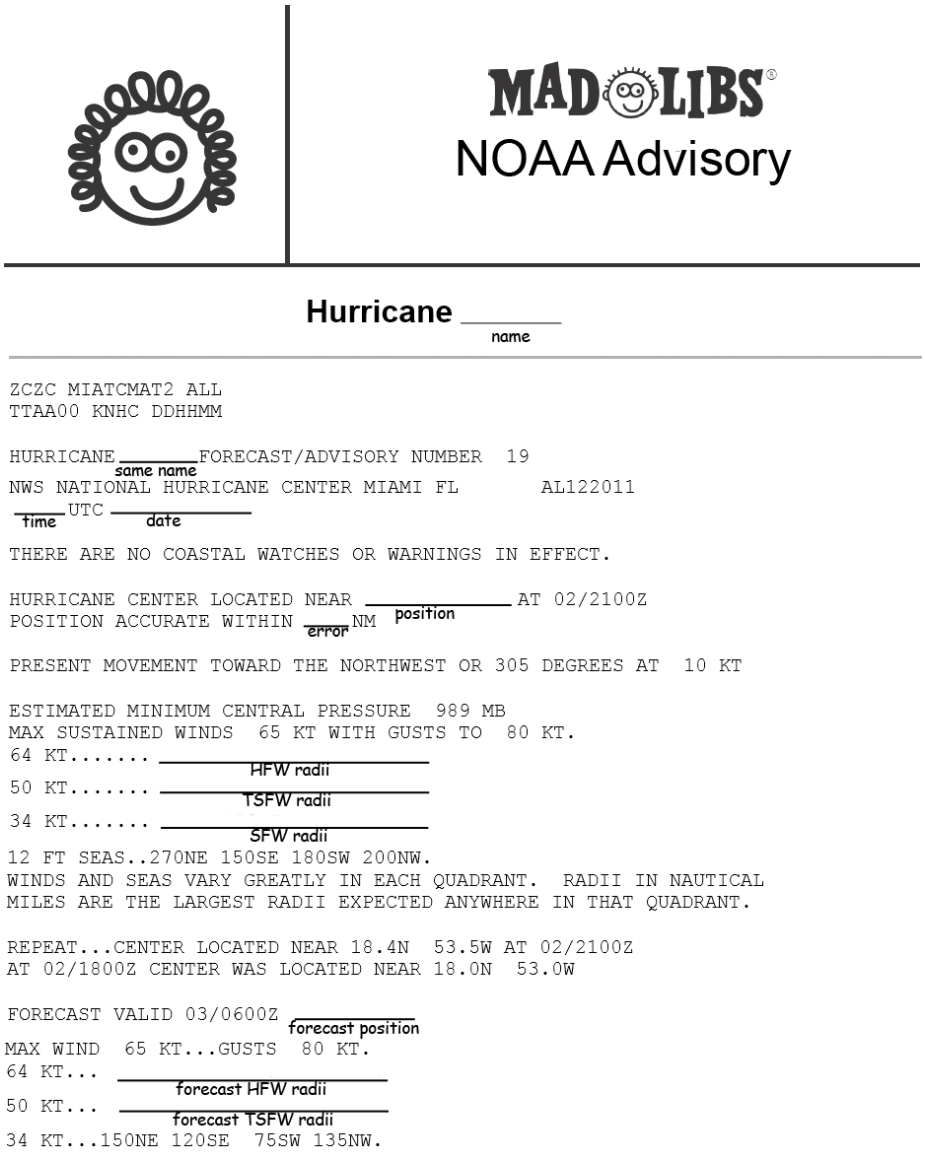

Figure 2: NOAA Advisory Mad Libs

(e) In between the quotation marks you should put the web address for the advisory archive of your hurricane.

6. `OUTPUT_SPREADSHEET_FILE = "" #...` This instruction is essentially the same as above, only with a different variable. This string will tell the script where to save the spreadsheet it makes, and what to call it. This is the **path** of the file, like a URL for a file on your own computer. When you've run tools in ArcGIS maybe you've noticed that after selecting a file or output, you're left with something like

   "**Output Feature Class:** `F:\GIS\lab_1\oberlin_dem.tif`".

   This is a file path, directions to a specific file or folder. In this example, the file "`oberlin_dem.tif`" is in the folder "`lab_1`" which is in the folder "`GIS`", etc.

7. `advisory_informaton_list = list()` This instruction both sets a variable, and calls a tool, or **function**.

   (a) It calls the "list" function, which **returns** an empty list. Saying a function "returns" something is a fancy way of saying it **evaluates** to something. You can think of $\sqrt{4}$ *evaluating* to the value 2, or as the square-root function *returning* the value 2.

   (b) Then it labels the empty list (the result of the list function) as "`advisory_information_list`

8. `advisory_url_list = fp.get_advisory_urls(ADVISORY_ARCHIVE_URL)` Another instruction that calls a function and then assigns the result to a variable. Most instructions in your Python scripts will be like this. `fp.get_advisory_urls` calls a function that returns a list of web addresses for each hurricane advisory. The input is the web address of the advisory archive for your hurricane, which was labeled as "`ADVISORY_ARCHIVE_URL`"

9. `for url in advisory_url_list:` This instruction is a little different from what we've seen so far. What it does is a create a loop, specifically a **for loop**, allowing for the same steps to be done multiple times. The steps that will be "looped over" are indented below the for loop.

   But how many times will this loop run? Notice a familiar label in the instruction that creates the loop, `advisory_url_list`. It will run as many times as there are items in the list that the variable `advisory_url_list` **references**. But what does `url` mean? It is in fact a variable that will be set each time the loop is run. It will, in turn, reference each entry in the list. The instruction `for url in advisory_url_list:` can be read as "for each item in the list labeled `advisory_url_list`, give the item the label `url` and then execute the following instructions.

10. `#...` Next we have two comments, ignored by the computer.

11. `advisory_function = fp.get_advisory_dictionary(url)` Notice that this line, as well as the line below it, is indented. This means that they are instructions that will be looped over. This is another instruction that calls a function and then sets a variable to reference the result of the function. This function, `fp.get_advisory_dictionary` gets information from a hurricane advisory. The function is given the URL of an advisory as input. `url` references a web address in the list that `advisory_url_list` refers to.

12. `advisory_information_list.append(advisory_information)` This instruction executes a function. This function adds the item referenced by the label `advisory_information` (set in the instruction above) to the list referenced by `advisory_information_list`. This is a special function specific to the list `advisory_information_list`, which is why the only input that the function takes is what is being appended to the list, not which list the item should be added to. Python automatically creates this function for every list.

13. `fp.advisory_list_to_csv(...)` This instruction also calls a function, which creates a spreadsheet from information. The function is given two things as input, a list to create the spreadsheet from, and a file path to save it to.

Now you understand what the script does. Fill in your own values for `ADVISORY_ARCHIVE_URL` and `OUTPUT_SPREADSHEET`. Then save the script and run it by clicking on the "Run" menu and then selecting "Run Module", or by hitting the "F5" key.