Assignment 5

Objectives: Practice working using strings and files, as well as their methods. Practice the Pythonic way to make `for` loops and to handle lists. Learn basic procedures to use csv files. Use the spiral approach to problem solving: solve a simplified version of the problem and then a more complex version; contnue increasing the complexity until the target problem has been solved.
Note: Include DocStrings in each script you submit analogous to the following:

```
"""This script characterizes squirrel-human interactions in NY.


Submitted by Mauricio Arias, NetID ma6918
[Include a short explanation here of how the script does that.]
"""
```

Background. Searching in csv files. Simple version.
In the attached file about 3000 interactions with squirrels were documented by a group of volunteers. Use that file to test the script described below.

In Figure 1, the file is shown as a spreadsheet. Notice the structure in rows and columns. Notice also that the first row contains the labels for the columns. Importantly, each row corresponds to a single encounter with a squirrel and conveys different types of information about it: fur color, location, activities the squirrel performed at the time of the encounter, etc. (Not all the columns are shown.)

In Figure 2, the file is shown as simple text. This reflects more closely how it is stored in the computer. Notice that there are many lines. The first line contains the labels for the "columns," which are not evident anymore but are there logically as inferred from Figure 1. This line (or "row") is said to be *the header row*. Importantly, the different labels in the first line are separated by a comma. The same is true for the elements of information in the other lines. This type of file is known as a comma-separated values, or *csv*, file. (The extension is .csv.) However, the separator does not need to be a comma: it could be a tab (\t) or a space among other options.

When reading the files, we will use the following pythonic template (use better names for the variables):

```
header = file_handle.readline()  # This reads and stores the header line.

for line in file_handle: …  # This for loop reads one line at a time.
```

The split() method

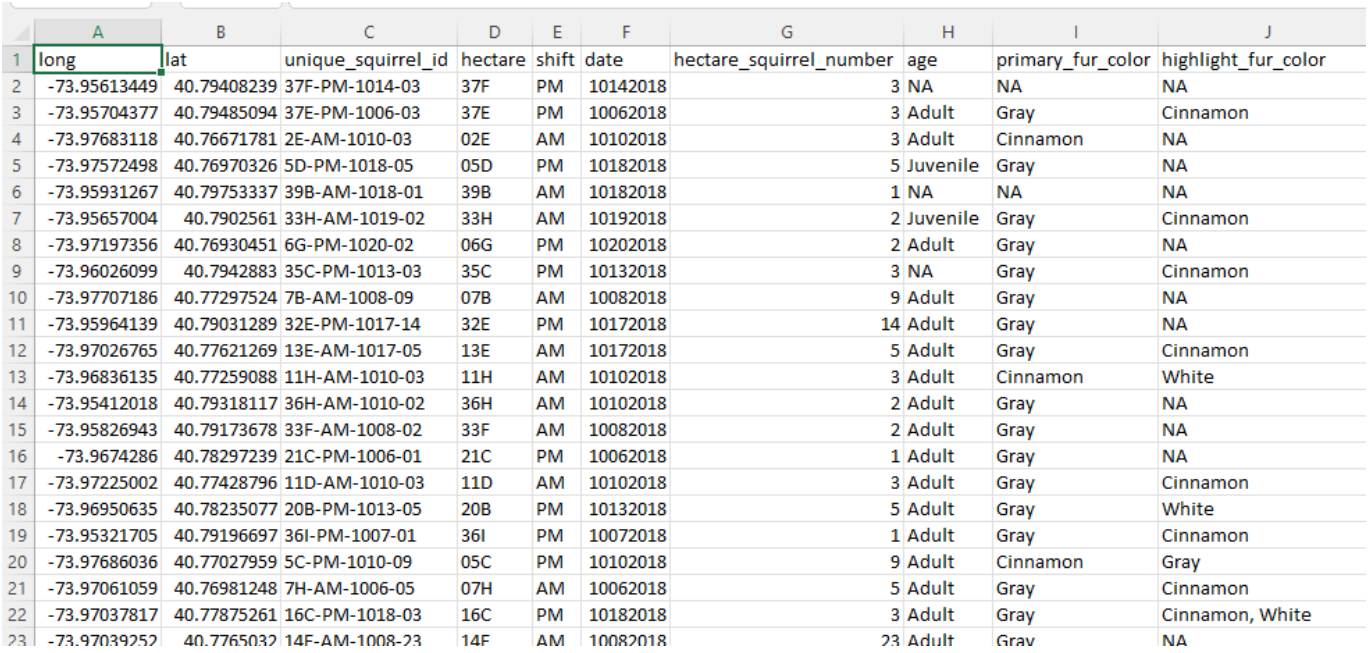
This method allows the separation of a string into several parts as specified by a separator.

```
details_list = encounter_info.split(",")  # encounter_info is a string.
```

The join method

This method is the converse of the split method but it is somewhat counter-intuitive in its use. Let's assume we have a list of strings: details_list. (It has to be strings.) The following statement generates a string (concatenated_details) with all the details separated by commas.

concatenated_details = ",".join(details_list)



	A	B	C	D	E	F	G	H	I	J
1	long	lat	unique_squirrel_id	hectare	shift	date	hectare_squirrel_number	age	primary_fur_color	highlight_fur_color
2	-73.95613449	40.79408239	37F-PM-1014-03	37F	PM	10142018	3	NA	NA	NA
3	-73.95704377	40.79485094	37E-PM-1006-03	37E	PM	10062018	3	Adult	Gray	Cinnamon
4	-73.97683118	40.76671781	2E-AM-1010-03	02E	AM	10102018	3	Adult	Cinnamon	NA
5	-73.97572498	40.76970326	5D-PM-1018-05	05D	PM	10182018	5	Juvenile	Gray	NA
6	-73.95931267	40.79753337	39B-AM-1018-01	39B	AM	10182018	1	NA	NA	NA
7	-73.95657004	40.7902561	33H-AM-1019-02	33H	AM	10192018	2	Juvenile	Gray	Cinnamon
8	-73.97197356	40.76930451	6G-PM-1020-02	06G	PM	10202018	2	Adult	Gray	NA
9	-73.96026099	40.7942883	35C-PM-1013-03	35C	PM	10132018	3	NA	Gray	Cinnamon
10	-73.97707186	40.77297524	7B-AM-1008-09	07B	AM	10082018	9	Adult	Gray	NA
11	-73.95964139	40.79031289	32E-PM-1017-14	32E	PM	10172018	14	Adult	Gray	NA
12	-73.97026765	40.77621269	13E-AM-1017-05	13E	AM	10172018	5	Adult	Gray	Cinnamon
13	-73.96836135	40.77259088	11H-AM-1010-03	11H	AM	10102018	3	Adult	Cinnamon	White
14	-73.95412018	40.79318117	36H-AM-1010-02	36H	AM	10102018	2	Adult	Gray	NA
15	-73.95826943	40.79173678	33F-AM-1008-02	33F	AM	10082018	2	Adult	Gray	NA
16	-73.9674286	40.78297239	21C-PM-1006-01	21C	PM	10062018	1	Adult	Gray	NA
17	-73.97225002	40.77428796	11D-AM-1010-03	11D	AM	10102018	3	Adult	Gray	Cinnamon
18	-73.96950635	40.78235077	20B-PM-1013-05	20B	PM	10132018	5	Adult	Gray	White
19	-73.95321705	40.79196697	36I-PM-1007-01	36I	PM	10072018	1	Adult	Gray	Cinnamon
20	-73.97686036	40.77027959	5C-PM-1010-09	05C	PM	10102018	9	Adult	Cinnamon	Gray
21	-73.97061059	40.76981248	7H-AM-1006-05	07H	AM	10062018	5	Adult	Gray	Cinnamon
22	-73.97037817	40.77875261	16C-PM-1018-03	16C	PM	10182018	3	Adult	Gray	Cinnamon, White
23	-73.97039252	40.7765032	14F-AM-1008-23	14F	AM	10082018	23	Adult	Gray	NA

Fig. 1. Screenshot of the file viewed as a spreadsheet. Notice the table-like structure it has: rows with many types of information about each encounter and columns showing the same information for different encounters.

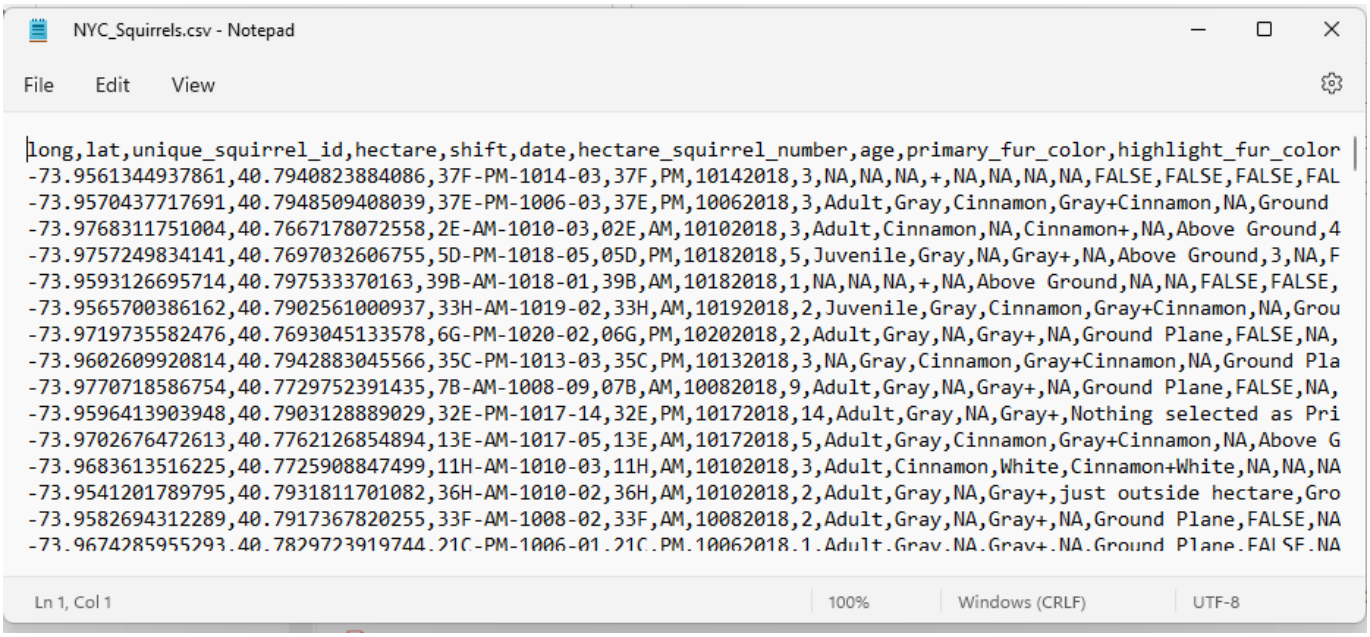


Fig. 2. The same file opened with a text-based editor: Notepad in this case. Notice that each value in a row is separated by a comma from the one that follows.

Part 1. *Squirrel!*

Write a script that searches for specific strings in a file. This script requests a text file name. If no filename is provided it exits. Otherwise the script requests a query string. It opens the file and processes the header into a list of labels. Remember to strip the line just read of the "\n" at the end! (Essentially every line you read will have this character at the end.)

After processing the header, the script presents the options and requests which column should be searched. The column number is validated. The following counts are made: lines with the query anywhere, total occurrences of the query in the file and total number of lines with the query in the requested column. The script then closes the file and reports the results. The script then goes back to asking the user which file it wants to search. (Notice that a single line can contain more than one occurrence of the query.)

After you are done programming the script, use it to analyze the encounters as described below. Include a results file made manually with the answers to the following queries:

1) For age: Adult, adult and Juvenile. Notice that the same result should be obtained irrespective of case.

2) For primary fur color: cinnamon, gray, grey, black, brown and white

3) For highlight fur color: cinnamon, gray, grey, black, brown and white

Manually save your answers as a simple text file (.txt) and name it according to the instructions below.

Submit your script as `[NetID]_characterizing_squirrels.py`. Submit your answers as `[NetID]_squirrels_chars.txt`. (5 pts)

Part 2. *Taxi!*

Imagine we are writing a script for someone trying to study how the pandemic affected people in the city. As part of their research they will like to use the for-hire vehicle trips as a proxy for activity. We will write for them a script that tallies trips according to data the city has collected.

	A	B	C	D	E	F	G	H
1	dispatching_base_num	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	SR_Flag	Affiliated_base_number	
2	B00014	2021-03-01 00:23:11	2021-03-01 02:45:00				B00014	
3	B00021	2021-03-01 00:39:16	2021-03-01 00:50:37	70	173		B00021	
4	B00021	2021-03-01 00:51:22	2021-03-01 01:01:55	129	82		B00021	
5	B00021	2021-03-01 00:42:07	2021-03-01 00:49:53	173	82		B00021	
6	B00021	2021-03-01 00:04:23	2021-03-01 00:32:56	198	198		B00021	
7	B00021	2021-03-01 00:50:56	2021-03-01 01:04:00	82	82		B00021	
8	B00037	2021-03-01 00:24:25	2021-03-01 00:36:03		177		B00037	
9	B00037	2021-03-01 00:56:09	2021-03-01 01:08:02		225		B00037	
10	B00037	2021-03-01 00:11:41	2021-03-01 01:01:23		151		B02878	
11	B00037	2021-03-01 00:03:10	2021-03-01 00:08:53		89		B00037	
12	B00037	2021-03-01 00:21:33	2021-03-01 00:43:49		65		B00037	
13	B00037	2021-03-01 00:48:54	2021-03-01 00:54:06		49		B00037	
14	B00053	2021-03-01 00:01:00	2021-03-01 02:03:00				B00053	
15	B00111	2021-03-01 00:15:00	2021-03-01 00:49:00				B00111	
16	B00112	2021-03-01 00:23:59	2021-03-01 00:24:27		14		B00112	
17	B00149	2021-03-01 00:02:52	2021-03-01 00:29:05		35		B00149	
18	B00149	2021-03-01 00:46:28	2021-03-01 01:00:05		61		B00149	
19	B00221	2021-03-01 00:10:26	2021-03-01 00:19:12		42		B00221	
20	B00221	2021-03-01 00:14:41	2021-03-01 00:16:32		243		B00221	
21	B00221	2021-03-01 00:18:45	2021-03-01 00:27:30		243		B00221	

Fig 3. For-hire vehicle trip data for the month of January 2021. Only part of the data is shown.

The script requests a day of the month and two filenames. It then opens the files and for the day specified, it groups and counts all the trips that occurred between the beginning of the hour and the end of the hour: each file is handled separately. For example, all the trips that started between 9:00:00 and 9:59:59 will be grouped together and the trips counted for each file. Another grouping will be made for the trips between 10:00:00 and 10:59:59 and the trips counted. There will be 24 groupings total for each file with their respective counts. (You only need to keep track of the counts.)

The script will then request the name of a file to save the results. It will open it for writing. The results will be provided a header with the day month and year for which the tally is provided. For example, it will say June 15 2021 in one of the columns and May 15 2022 for the other. This of course depends on the requested day and the files provided. (Notice that it has to deduce the year and deduce and properly write the name of the month.)

Use the script to count all the trips starting on Valentine's day in 2020 and separately in 2021. Tally them by the hour. Download appropriate files with the records for For-Hire Vehicle trips from the nyc.gov website.

Use the join method to save the results as comma separated values.

Submit your script as `[NetID]_counting_trips.py` and the results as a csv file `[NetID]_taxi_results.csv`. (5 pts)