Homework 7: Regexes and Structured Data

Due Thursday, November 14, 11:59 pm

Worth 35 points

Read this first. A few things to bring to your attention:

1. We will be using the Cavium cluster in the next homework, so now is the time to verify that you have a username on the cluster. If you have not already done so, please sign on to the cluster by following the instructions at `https://arc-ts.umich.edu/cavium/user-guide/#document-3`. Contact the professor or your GSI if you are unable to connect to the cluster.

2. Start early! If you run into trouble installing things or importing packages, it's best to find those problems well in advance, not the night before your assignment is due when we cannot help you!

3. **Make sure you back up your work!** I recommend, at a minimum, doing your work in a Dropbox folder or, better yet, using `git`, which is well worth your time and effort to learn.

4. **Be careful to follow directions!** Remember that Python is case sensitive. If we ask you to define a function called `my_function` and you define a function called `My_Function`, you will not receive full credit.

5. **A note on grading:** overly complicated solutions or solutions that suggest an incomplete grasp of key concepts from lecture will not receive full credit.

Instructions on writing and submitting your homework.

Failure to follow these instructions will result in lost points. Your homework should be written in a jupyter notebook file. I have made a template available on Canvas, and on the course website at `http://www-personal.umich.edu/~klevin/teaching/Fall2019/STATS507/hw_template.ipynb`. You will submit, via Canvas, a `.zip` file called `yourUniqueName_hwX.zip`, where `X` is the homework number. So, if I were to hand in a file for homework 3, it would be called `klevin_hw3.zip`. Contact the instructor or your GSI if you have trouble creating such a file.

Please include all code necessary to run all problems in the homework in your submission unless instructed otherwise. Please include in your notebook file a list of any and all people with whom you discussed this homework assignment as well as estimates of how many hours you spent on each of the sections of the assignment.

Detailed instructions on submitting your homework can be found on the course web page at `http://www-personal.umich.edu/~klevin/teaching/Fall2019/STATS507/hw_instructions.html`. Please direct any questions to either the instructor or your GSI.

1 Regular Expressions: Warmup (8 points)

In this problem, you'll get practice with basic regular expressions. Pay particular attention to edge cases such as the empty string and single-character strings when writing your regexes. At the URL `http://www.greenteapress.com/thinkpython/code/words.txt` is a list of about 100,000 English words.

1. Use `urllib` to open the URL and read the file, and produce a list of ASCII strings so that each line of the file corresponds to an element of the list. You will likely need to convert the raw bytes read from the webpage to ASCII characters, for which you should see the documentation for the string methods `encode` and `decode`. How many words are in the file?

2. It is a good habit to always look at your data to check that it makes sense. Have a look at the words in the list. Does anything jump out at you? **Note:** I am not requiring you to do anything specific, here. Just look at the data!

3. Write a regular expression that matches any string containing exactly three consecutive vowels. Compile this regular expression, and assign it to a variable called `three_consecutive_vowels`. Use this regex to determine how many words from the list contain exactly three consecutive vowels. For the purposes of this **specific** subproblem, the vowels are `a, e, i, o, u`. All other letters are consonants. Produce a list of all such words.

4. Write a regular expression that matches any string that contains no instances of the letter `e`. Compile this regular expression, and assign it to a variable called `gadsby`. (*Gadsby* is the title of an English novel written in the 1930s that contains *almost* no instances of the letter `e`). How many words in the list do not contain the letter `e`?

5. Write a regular expression that matches any string that begins and ends with a consonant and has no consonants in between. For the purposes of this **specific** subproblem, `y` is neither consonant nor vowel, so consonants are the 20 letters that are not one of `a, e, i, o, u, y` and vowels are `a, e, i, o, u`. The words need not begin and end with the *same* consonant, so `door` is a valid match. Compile this regular expression, and assign it to a variable called `consonant_consonant`. How many words begin and end with a consonant with no consonants in between?

6. Write a regular expression that matches any string whose last two characters are the first two characters in reverse order. So, for example, your regex should match `repeater` and `stats`, but not `neoprene`. Compile this regular expression and assign it to a variable called `bookends`. How many words in the list have this property? **Hint:** be careful of the cases in which the word is length less or equal to 3. You may handle the case of a single character (e.g., `a`), as you like, but please give an explanation for your choice.

2 Exploring Internet Traffic with Regexes (7 points)

In this problem, you'll get a taste of a more realistic application of regular expressions. The file `http://umich.edu/~klevin/teaching/Fall2019/STATS507/SkypeIRC.txt` contains data generated by web traffic associated with Skype and IRC, captured using the Wireshark program, a common tool for analyzing web traffic. The original data file

can be found on the Wireshark wiki, `https://wiki.wireshark.org/SampleCaptures`, but please use the file provided on my website for this assignment.

1. Download the file from the URL above (or use `urllib` or `requests` to open it directly, being careful to convert the raw bytes back to UTF-8) and read its contents into a string. Each line of this file corresponds to a single packet sent over the internet. How many packets are in this file? Save the answer in a variable `n_packets`. **Note:** if you decide to download the file, don't forget to include a copy of it in your submission so that we can run your code.

2. Use regular expressions to extract all the IP addresses from the file and collect them in a Python list. An IP address consists of four numbers, which are displayed as `A.B.C.D` where `A,B,C` and `D` are each numbers between 0 and 255 inclusive. How many unique IP addresses appear in the data set? Save the answer in a variable `ip_addresses`. **Note:** there are a few pieces of text in the file that *look* like IP addresses but aren't (e.g., they are of the form `A.B.C.D`, but one or more of `A`,`B`,`C` and `D` are outside the range 0 to 255). You should try to craft a regex that matches as few of these as you can, but of course it will not be possible to avoid certain edge cases.

3. Write a function called `get_packets_by_regex` that takes a single raw string as its argument and returns all lines of the input file that match the input raw string as a regular expression. So, for example, `get_packets_by_regex(r'comcast')` will return all lines from the file containing the string `'comcast'`. Your function should perform appropriate error checking to ensure that the input is a string, but you do not need to check that it is a raw string.

4. The second piece of text (i.e., non-whitespace) on each line is a time stamp, counting the time (in seconds) since the beginning of the traffic recording. Using `matplotlib`, create a plot displaying how many packets appeared in each second of the recording. A histogram or line plot is the most obvious way to do this, but you should feel free to use a more creative way of displaying this information if you wish to do so. Save your plot as a PDF, in a file called `timestamps.pdf` and include this file in your submission. **Note:** in case it wasn't obvious, there is no need to use a regular expression for this subproblem if you do not want to.

3 Retrieving Data from the Web (10 points)

In this problem, we'll scrape data from Wikipedia using `BeautifulSoup`. Documentation for BeauitfulSoup can be found at `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`. As mentioned in lecture, there is another package, called `requests`, which is becoming quite popular, which you are welcome to use for this problem instead, if you wish. Documentation for the `requests` package can be found at `http://docs.python-requests.org/en/master/`.

Suppose you are trying to choose a city to vacation in. A major factor in your decision is weather. Conveniently, lots of weather information is present in the Wikipedia articles for most world cities. Your job in this problem is to use `BeautifulSoup` to retrieve weather information from Wikipedia articles. We should note that in practice, such information is typically more easily obtained from, for example, the National Oceanic and Atmospheric

Administration (NOAA), in the case of cities in the United States, and from analogous organizations in other countries.

1. Look at a few Wikipedia pages corresponding to cities. For example:

   - `https://en.wikipedia.org/wiki/Ann_Arbor,_Michigan`
   - `https://en.wikipedia.org/wiki/Buenos_Aires`
   - `https://en.wikipedia.org/wiki/Harbin`

   Note that most city pages include a table titled something like "Climate data for [Cityname] (normals YYYY-YYYY, extremes YYYY-YYYY)" Find a Wikipedia page for a city that includes such a table (such as one of the three above). In your jupyter notebook, open the URL and read the HTML using either `urllib` or `requests`, and parse it with `BeautifulSoup` using the standard parser, `html.parser`. Have a look at the parsed HTML and find the climate data table, which will have the tag `table` and will contain a child tag `th` containing a string similar to

   ```
   Climate data for [Cityname] (normals YYYY-YYYY, extremes YYYY-YYYY).
   ```

   Find the node in the `BeautifulSoup` object corresponding to this table. Describe the structure of this node of the tree (e.g., how many children does the table have, what are their tags, etc. A sentence or two is fine, here). You may want to learn a bit about the structure of HTML tables by looking at the resources available on these websites:

   - `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/table`
   - `https://www.w3schools.com/html/html_tables.asp`
   - `https://www.w3.org/TR/html401/struct/tables.html`

2. Write a function `retrieve_climate_table` that takes as its only argument a Wikipedia URL, and returns the `BeautifulSoup` object corresponding to the climate data table (if it exists in the page) and returns `None` if no such table exists on the page. You should check that the URL is retrieved successfully, and raise an error if `urllib2` fails to successfully read the website. You may notice that some city pages include more than one climate data table or several nested tables (see, for example, `https://en.wikipedia.org/wiki/Los_Angeles`). In this case, your function may arbitrarily choose one of the tables to return as a `BeautifulSoup` object. **Note:** a good way to check for edge cases is to test your script on the Wikipedia pages for a few of your favorite cities. The pages for Los Angeles, Hyderabad and Boston will give good examples of edge cases that you should be able to handle, but note that these are by no means exhaustive of all the possible edge cases. **Hint:** make use of the `contents` attribute of the `BeautifulSoup` objects and the ability to change the elements of the contents list to Unicode.

3. As you look at some of the climate data tables, you may notice that different cities' tables contain different information. For example, not all cities include snowfall data. Write a function `list_climate_table_row_names` that takes as its only argument a Wikipedia URL and returns a list of the row names of the climate data table, or returns `None` if no such table exists. The list returned by your function should, ideally, consist solely of Python strings (either Unicode or ASCII), and should not include any `BeautifulSoup` objects or HTML (**Hint:** see the `BeautifulSoup` method

`get_text()`). The list returned by your script should *not* include an entry corresponding to the `Climate data for...` row in the table. **Second hint:** you are looking for HTML table header (`th`) objects. The HTML attribute `scope` is your friend here, because in the context of an HTML table it tells you when a `th` tag is the header of a row or a column.

4. The next natural step would be to write a function that takes a URL and a row name and retrieves the data from that row of the climate data table (if the table exists and has that row name). Doing this would require some complicated string wrangling to get right, so I'll spare you the trouble. Instead, please **briefly** describe either in pseudo code or in plain English how you would accomplish this, using the two functions you wrote above and the tools available to you in the `BeautifulSoup` package. **Note:** just to be clear, you **do not** have to write any Python code for this last step.

4 Relational Databases and SQL (10 points)

In this problem, you'll interact with a toy SQL database using Python's built-in `sqlite3` package. Documentation can be found at `https://docs.python.org/3/library/sqlite3.html`. For this problem, we'll use a popular toy SQLite database, called Chinook, which represents a digital music collection. See the documentation at

`https://github.com/lerocha/chinook-database/blob/master/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite`

for a more detailed explanation. We'll use the `.sqlite` file `Chinook_Sqlite.sqlite`, which you should download from the GitHub page above. **Note:** Don't forget to save the file in the directory that you're going to compress and hand in, and make sure that you use a relative path when referring to the file, so that when we try to run your code on our machines the file path will still work!

1. Load the database using the Python `sqlite3` package. How many tables are in the database? Save the answer in the variable `n_tables`.

2. What are the names of the tables in the database? Save the answer as a list of strings, `table_names`. **Note:** you should write Python `sqlite3` code to answer this; don't just look up the answer in the documentation!

3. Write a function `list_album_ids_by_letter` that takes as an argument a single character and returns a list of the primary keys of all the albums whose titles start with that character. Your function should ignore case, so that the inputs "a" and "A" yield the same results. Include error checking that raises an error in the event that the input is not a single character.

4. Write a function `list_song_ids_by_album_letter` that takes as an argument a single character and returns a list of the primary keys of all the songs whose album names begin with that letter. Again, your function should ignore case and perform error checking as in `list_album_ids_by_letter`. (again ignoring case). **Hint:** you'll need a JOIN statement here. Don't forget that you can use the `cursor.description` attribute to find out about tables and the names of their columns.

5. Write a function `total_cost_by_album_letter` that takes as an argument a single character and returns the cost of buying every song whose album begins with that letter. This cost should be based on the tracks' unit prices, so that the cost of buying a set of tracks is simply the sum of the unit prices of all the tracks in the set. Again your function should ignore case and perform appropriate error checking.