When searching for websites in a search engine, pages from a wide range of diverse areas may be returned, e.g., topics from mathematics to politics to weather to sports. A well-known website that organizes a wide variety of these topics is Wikipedia, the online collaborative encyclopedia.

This assignment is on analysing data from a site hosting over 500 pages taken from Wikipedia.

The two seed links that need to be worked on are:

- http://115.146.93.142/fullwiki/A12_scale
- http://115.146.93.142/fullwiki/Gerard_Maley

Task 1: Get all links

Implement the function task1(…) in task1.py that takes two arguments:
- The first argument is a list of starting links in the format of a list of strings and
- The second argument is a filename.

The function should crawl each starting link and output a json formatted output file with the filename given in the second argument.

Task 2: Extracting words from a Page

For this task BeautifulSoup should be used. Each page will contain a div with the id of mw-content-text, which processing will be restricted to. You can use inspect element tool on one of the pages (or through Wikipedia itself) to see what this typically covers. Implement the function task2(..) in task2.py which takes a URL and a filename and outputs a json formatted file with that filename. The format of the JSON object output should be the name of the URL as the string and the value should be a list of words. If the page cannot be retrieved, the list should contain no values.

Two stages of pre-processing should occur, the first narrowing down and removing all irrelevant elements from the page, and the second which produces word tokens and removes irrelevant tokens. The first stage comprises eight steps:

1. For the rest of the steps, operate on the `div` element with `id` of `mw-content-text` discarding other parts of the page.
2. From the remaining page tree, remove all `th` elements with the `class` of `infobox-label`.
3. From the remaining page tree, remove all `div` elements with the `class` of `printfooter`.
4. From the remaining page tree, remove all `div` elements with the `id` of `toc`.
5. From the remaining page tree, remove all `table` elements with the `class` of `ambox`.
6. From the remaining page tree, remove all `div` elements with the `class` of `asbox`.
7. From the remaining page tree, remove all `span` elements with the `class` of `mw-editsection`.
8. From the remaining `mw-content-text` tree, extract the text from the page using BeautifulSoup, this extraction should use *a space separator* (`' '`) between

elements, and should **not** connect text from adjacent elements. For example if a section of the remaining tree contents were <p>a</p><p>b</p> it should become `'a b'`, not `'ab'`.

After the first stage's step 1 -- 8 have been completed, the remaining text should be converted to tokens in the following steps

1. Change all characters to their `casefold`ed form and then normalize all page text to its `NFKD` form.
2. Convert all non-alphabetic characters (for example, numbers, apostrophes and punctuation), except for spacing characters (for example, whitespaces, tabs and newlines) and the backslash character (`'\'`) to single-space characters. For example, `'&'` should be converted to `' '`. You should consider non-English alphabetic characters as non-alphabetic for the purposes of this conversion.
3. Convert all spacing characters such as tabs and newlines into single-space characters, and ensure that only one whitespace character exists between each token.
4. The text should then be converted to explicit tokens. This process should use single-space characters as the boundary between tokens.
5. Remove all stop words in `nltk`'s list of English stop words from the tokens in the page text.
6. Remove all remaining tokens that are less than two characters long from the page text.
7. Each token should then be converted to its Porter stemming algorithm stemmed form. This should use all the default improvements present in the NLTK PorterStemmer (including both the additional improvements beyond the original algorithm and NLTK's improvements).

Once steps 1 -- 7 are done, build a JSON file representing the page. The JSON file should contain a JSON object containing one string matching the fully qualified url requested and the associated value should be an array of strings corresponding to the tokens produced by the above pre-processing.

Task 3: Producing a bag of words for All Pages

Implement the function task3(..) in task3.py which takes a dictionary containing key/value pairs matching the output format of Task 1 and a filename and outputs a CSV formatted file with that filename.

For each key in the dictionary, all links should be visited. The key should be considered the seed_url and the link should be considered the link_url. The HTML page for each link_url should be retrieved and processed into a list of tokens according to both stages of pre-processing in Task 2, this list of tokens should then be joined with a single space and form the words string associated with the page. Once all pages are processed, a dataframe should be created which is output to a new csv file with the given filename. The csv output should contain the headings link_url, words, seed_url and each row should contain the details of one page, with

- link_url being the fully qualified URL of the page the bag of words corresponds to,
- words being a string comprising all tokens for the link_url's page, separated by exactly one space character and
- seed_url being the key which the link URL was taken from in the dictionary argument.

If no tokens are retrieved after pre-processing, the words string should be an empty string ''.

The rows in the csv file should be in ascending order of link_url and then (where the link_url is the same for two or more pages) seed_url.

Task 4: Plotting the most Common Words

Implement the function task4(..) in task4.py which takes a pandas dataframe matching the output format of Task 3 and one filename and outputs a plot to that filename. task4(..) should return a dictionary where each seed_url is a key and the value is the list of 10 most common words.

In this task you should generate a plot which should allow the comparison of the top 10 most common words in each seed_url . Here the metric of number of total occurrences across all pages should be used (e.g. if one page has two uses of a word, both should be counted in the total for the seed_url). For consistency in your output, if multiple words appear equally frequently, they should appear in alphabetical order in your plot.

Task 5: Dimensionality Reduction

Implement the function task5(..) in task5.py which takes one pandas dataframe matching the output format of Task 3 and two filenames and outputs a plot to each of the given filenames.

In task5(..) you should first produce a bag of words representation of the words across all pages and then perform normalisation using sklearn's max norm fit to the dataframe, following this, you should perform Principal Component Analysis using 2 components to the normalised dataframe.

task5() should output two files:

- The plot output to the first file should show the top 10 most positively weighted tokens and their weights and 10 most negatively weighted tokens and their weights for each component.
- The plot output to the second file should show where articles from each seed_url fall on the 2-component axes.

task5() should return a dictionary where the PCA component ID is the key (0 for the component explaining the greatest variance and 1 for the component explaining the next greatest variance) and the value is also a dictionary, where this nested dictionary has four keys, "positive", "negative", "positive_weights", "negative_weights" and the associated value of each key is as follows:

- The value for the positive key should be a list of the top 10 tokens with highest magnitude positive weights.
- The value for the `negative` key should be a list of the top 10 tokens with highest magnitude negative weights.
- The value for the positive_weights key should be the weight in the PCA decomposition of the relevant token in the positive token list.
- The value for the negative_weights key should be the weight in the PCA decomposition of the relevant token in the negative token list.
-