CSC 242 Section 504 Winter 2017
Homework 7: Extracting headlines again

Due date as specified on D2L

This assignment is worth 2% of your overall grade, and therefore will be graded on a scale of 0-2. Please upload a file containing your competed code onto D2L by the due date. While you may discuss the assignment with others, **you must write your code by yourself,** without assistance from other students or anyone else. Please see the course syllabus for details, and for my late homework submission policy.

Headline collecting

In a previous lab, we collected headlines in a "brute force" fashion, by performing string search for likely headline start tags, any embedded HTML elements, etc. This time, we will perform the same task using the HTMLParser tool. When properly working, the program should be given a URL, and print (to the console) all of the headlines on the page.

In order to do this, we will need to write all 3 handler methods for HTMLParser. Here is a brief description of what each should do:

1. handle_startag: Checks the tag to see if it is a headline tag (<h1>, <h2>, or <h3>). We will not concern ourselves with a threshold for this assignment. If the tag is a headline tag, then a flag is set to True in order to indicate that a headline element has been found. This is necessary for the handle_data method to extract the headline.
2. handle_data: Check the flag set by handle_starttag to see if we are in a headline element. If so, the the headline should be printed.
3. handle_endtag: Sets the flag back to False, indicating that we are no longer in a headline element.

What I am providing

I have provided a template for you to help you write your code.

Extra credit

For one point of extra credit, the program should write its output to a HTML file rather than to the console. In the HTML file, headline start and end tags should be included to indicate the size of the headline in the original document. For example, see headlines.html, which was generated by my sample solution from the New York Times homepage o 3/7 at 10:15 PM.

An absolute URL consists of a protocol (http://, https://, etc) followed by a host name (e.g., www.depaul.edu) and finally other information that enables the host to find the requested Web resource. For example, the following Web site provides helpful information about current traffic in the Chicago area:

http://www2.ai.uic.edu/lmiga/map.jsp?mapname=chicagoArea

The protocol is http://, the host is www2.ai.uic.edu, and the rest of the URL /lmiga/map.jsp?mapname=chicagoArea provides the host with enough information to find the map of Chicago traffic.

While use of absolute URLs is encouraged on all Web pages, many Web authors (including me) use "relative" URLs. These do not start with a protocol or host name, but assume that the absolute URL can be computed by combining the relative URL with the context in which it appears. If a relative URL is passed to the urlopen function, it will not succeed in opening the page; thus, the relative URL must first be converted to an absolute URL. For example, my condor homepage is

http://condor.depaul.edu/slytinen

On this page, I have a link to information about my research. The hyperlink is: Research information. Web browsers can figure out the absolute URL by assuming that "research" is a subfolder of the folder in which my homepage is located; thus the absolute URL is http://condor.depaul.edu/slytinen/research/research.html.

Fortunately, Python has a tool to compute an absolute URL from a relative one. The function is call urljoin. It is passed 2 parameters: an absolute URL and a relative URL. For example:

urljoin('http://condor.depaul.edu/slytinen/', 'research/research.html') returns 'http://condor.depaul.edu/slytinen/research/research.html'.

You will need to use urljoin to compute absolute URLs for some of the images that your program collects.

Example

When I ran the Crawler on the New York Times Web site on 3/8/2017 at 5 PM and wrote image elements to 'images.html', the program generated the attached file.