COM110
Python for the web

1) Web pages are written in their own language so that browsers have instructions for how to display text, images/video, tables, alignment, colors, positioning, etc.

a) This "language" is not a proper programming language, but is instead more of a type-setting specification, and is called HTML (Hypertext Markup Language).
b) If you go to any web page (like www.conncoll.edu) using a web browser (like Chrome), then right-click and choose "View page source" to view its source code, the browser will open a window showing you the html code of that web page. Try this to see what this looks like.
c) Notice that HTML is characterized by html "tags," which generally look like <xxx> and are often followed by </xxx> . For example, in order to put a specific title in the title bar of a browser window (for example, Conn's home page) you would write

```
<title> Connecticut College </title>
```

You should be able to look through or do a find (ctrl+f) to find the <title> tag in the page source of the www.conncoll.edu website.
d) The tags communicate to the browser how the text between the tags should be formatted.
e) The text to be formatted is contained between the beginning (`<xxx>`) and ending (`</xxx>`) tag. So anyone can write html code and then have the browser display it.
f) Try writing a simple html file using any plain text editor (like Notepad): start with a title (using the `<title>` tag) and then write some text in bold (`<b>` … `</b>`).
g) Make sure the file is in plain text, save the file with an .html ending and then open it with a web browser. Notice that the title you specified appears in the title bar of the browser window.

2) The address of a website is known as its URL. The modules we will use are downloaded from the Standard Python library online. They are: `urlparse` (to manipulate URL strings) and `urllib` (supporting communication with the standard Web protocols such as http and ftp).

a) Let's start by writing a module to go to a URL and look at the source code of the website. Open the module `urlTest.py` and test with the code. Verify that the imported `urlretrieve` method gets the source code of a URL, then saves it as a plain text file that gets automatically created in the folder where you are running the program. Try retrieving different web pages and saving them as text or html files. View the contents of the text file that gets created each time.
b) Add code to the `openUrl` method that will read the html file into a string variable (the same old `read`(), `readline`() and `readlines`() command work on the `urlFile` pointer). Name your string variable `webpagecode`. (**Careful: if you use the `read()` method, it actually will not return a string as we might expect. You'll have to cast is as a string using the `str()` function.**)

c) Try test-printing the string `webpagecode` to make sure it worked.

d) Now let's look for the title of the page in the string of html that we read into `webpagecode`. As we learned above, the title of the web page will come after the html tag `<title>` and before the end tag `</title>`.

e) Now see if you can find the location/index of the `<title>` tag in the string. (Hint: you can use the built-in string function `webpagecode.find()` that we learned on page 140 of Zelle. It will give you the location of the first character of the string you're searching for. The location will be an integer that represents the index within the string you're searching in.) Store the index in a variable, and then do the same with the end title tag `</title>`. Print out the values of these indices to make sure they make sense (e.g., the index of the end title tag should naturally come *after* the index of the start title tag).

f) Now that we've located the title of the web page, we can use these indices to slice it from the string of html code (`webpagecode`) and print it. (Don't hard-code these indices though.)

g) In addition, assign the title to a new instance variable of `URLReader`. Call it `self.title`. (You can initialize it to empty string in the constructor.)

3) Write another method for the `URLReader` class that is called `displayTitle` (no parameters other than `self`). This method will create a new web page that only has one sentence of text announcing the title of the web page that you already opened in the `openURL` method, and then it will display this new web page in a web browser.

a) First, using the standard file writing syntax we learned early in the semester (see the final checkpoint of lab 2), `open` an output file in "write" mode, giving it the name "sample.html".

b) Next, store the following into a string variable:

```
"<html> <body>The title of the page is <i>"+self.title+"</i> </body> </html>"
```

Note that the `<i>` and `</i>` tags make the text in between them italicized.

c) Write this string to the output file and then close up the output file. (You can test at this point by adding code that calls this new method, and making sure this file gets created. Double-click it to see what the web page looks like.)

d) Then, finally, call openBrowser() method to display the file.

```
self.openBrowser("sample.html")
```

4) Reading and processing data from the Web. The US Geological Survey (USGS) keeps a map of earthquakes at the URL http://earthquake.usgs.gov/earthquakes/map/ . The data can be found here: http://earthquake.usgs.gov/earthquakes/ . The plain text earthquake data can be found under the "Real Time Feeds" link toward the bottom in various formats. E.g., here is the data for the past 7 days of earthquakes of magnitude 2.5 or greater in json format: http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_week.geojson

a) If you go to think link you will see that the data is in rows, one to an earthquake. You can see this data much more clearly if you put the data in a text editor and make sure line wrapping is turned off so that each row of data is visually on one line. Mac's TextEdit doesn't let you do this, so you'll really need some other text editor for this.
b) Each row consists of quite a bit of information, including the latitude and longitude, region, date and time, magnitude of the earthquake, etc.
c) The items in each row are contained in brackets and separated by commas.
d) What we would like to do is read in this information and find the average (mean) magnitude of all the earthquakes over the last 7 days.
e) Open the module quake.py and note that initializes numQuakes to 0 and totMag to 0 and then begins a for loop.
f) Complete the loop by doing the following for each line that gets read in:
    i) First convert the line into a string using str(). [Again, the file read methods aren't working exactly as they did for regular file objects, so they aren't exactly returning strings to us.]
    ii) Find the position in the string of the magnitude, using a .find("mag") call.
    iii) Similarlyfindthepositionofthecommaimmediatelyfollowingthemagnitude.
    iv) Slice out the earthquake's magnitude.
    v) Use the eval() function to convert the magnitude from a string to a float,
    vi) Then add it to totMag.
    vii) Of course, don't forget to increment numQuakes.
g) Afterward, return the average earthquake magnitude (totMag/numQuakes).
h) In main(), use a descriptive sentence to output this average magnitude.

5) Write additional similar methods to be called so you can print out a more complete statistical report on the earthquakes from part 4. Include not just mean magnitude of the earthquakes, but also output:
a) the max earthquake magnitude
b) the min earthquake magnitude
c) regional frequency ranking (which region got most frequent earthquake?)
d) the median earthquake magnitude [*challenge*]
e) the mode earthquake magnitude [*challenge*]
f) etc