Getting-Data-II-Solutions

May 21, 2019

1 Getting Data from the Internet (Part 2)

Obtaining and processing data

Author: Dhavide Aruliah

1.0.1 Assignment Contents

- Section **??**
- Section **??**
  - Section 1.1
  - Section 1.1
  - Section 1.1
  - Section 1.1
  - Section 1.1
  - Section 1.1
- Section **??**
  - Section 1.1.1
  - Section 1.1.1
  - Section 1.1.1
  - Section 1.1.1
  - Section 1.1.1

EXPECTED TIME 1.5 HRS

1.0.2 Overview

You have seen how to obtain data programmatically from we services using the Python `requests` module. You have also seen how to process data retreived in some common web formats (JSON and XML) using appropriate Python libraries (`json` and `lxml` respectively.

In this assignment, you will focus on working with HTML retrieved from the web using the Python module `bs4` (which stands for *BeautifulSoup 4*). This module provides basic tools for web-scraping from HTML data.

The content here is drawn from Video lectures 8-1 through 8-9.

1.0.3 Activities in this Assignment

- Using `requests` to retrieve HTML pages
- Using the `BeuatifulSoup` class from `bs4` to capture the content of HTML data
- Using common methods and attributes (`find`, `find_all`, `get`, `text`, etc.)

1.0.4 Using `bs4`

The class `BeautifulSoup` from the module `bs4` provides all the functionality you'll need for this assignment. This is not exhaustive; the `selenium` module, for instance, can be required for scraping more sophisticated web pages.

We'll import the `requests` module again because that is a good resource for obtaining web content.

```
In [ ]: from bs4 import BeautifulSoup
        import requests
```

1.1 Scraping Monty Python Quotes

To start, download some web data using the Python `requests` module as you did in the previous assignment. The data consists of Monty Python quotations taken from AllGreatQuotes.com.

Section 1.0.1

Question 1 Your task is as follows:

- Obtain the text from the web page `URL_quotes` by extracting the `.text` attribute from an associated `Response` object. Assign the result to a string `text_quotes`. This text is the content of the page in HTML.
- Use the `str.find` method to extract the *title* from the associated HTML. This is the text that lies strictly between the tags `<title>` and `</title>`. Assign the result to a string `title_quotes`.

```
In [ ]: ### GRADED
        ### Use requests.get to query URL_quotes (provided)
        ### Extract the .text attribute from the response obtained and bind
        ###     the resulting string to the identifier text_quotes.
        ### Use the string method find to identify the text (strictly) between
        ###     the substrings "<title>" and "</title>" in text_quotes.
        ###     Assign the identifier title_quotes to that substring of text_quotes.
        URL_quotes = 'http://www.allgreatquotes.com/monty_python_quotes.shtml'

        text_quotes = requests.get(URL_quotes).text
        start = text_quotes.find('<title>')+7
```

```
stop = text_quotes.find('</title>')
title_quotes = text_quotes[start:stop]


### For verifying answer:
print('title_quotes: {}\n\n'.format(title_quotes))
print(text_quotes[:110])
```

Section 1.0.1

Question 2 Using Python string methods to extract data from HTML is inefficient and difficult. The `BeautifulSoup` class is designed to make this much easier.

Your task in this question is to parse the string `text_quotes` into a `BeautifulSoup` object. Use the `lxml` parser to decode the text. Assign the result to the identifier `soup_quotes`.

In [ ]:
```
### GRADED
### Instantiate a BeautifulSoup object using the string text_quotes from Question 1.
###     Use the 'lxml' parser in the call to BeautifulSoup.
###     Assign the result to soup_quotes.
###

soup_quotes = BeautifulSoup(text_quotes, 'lxml')


### For verifying answer:
print('type(soup_quotes): {}'.format(type(soup_quotes)))
```

Section 1.0.1

Question 3 Your task in this question is to repeat the computation from Question 1 using the method `find` associated with `BeautifulSoup` object `soup_quotes` constructed in Question 2. Recall the `find` method finds the first tag matching the given identifier in the `BeautifulSoup` object; in this case, you are looking to match the tag `'title'`.

Assign the result of the `find` command to the identifier `title_quotes`.

In [ ]:
```
### GRADED
### Use the find method to extract the next 'title' tag from the BeautifulSoup object
###    soup_quotes. Assign the result to title_quotes.

title_quotes = soup_quotes.find('title')


### For verifying answer:
print('title_quotes: {}\n'.format(title_quotes))
print('type(title_quotes): {}\n\n'.format(type(title_quotes)))
print(title_quotes.text)
```

Section 1.0.1

Question 4 The `BeautifulSoup` object has a `find_all` method that resembles the `find` method. The difference is that the result returned is a `ResultSet` object (which is basically a list of `bs4` elements).

In this question, you will extract all the tags from `soup_quotes` that match `'title'` (not just the first). Assign the result of the `find_all` command to the identifier `all_title_quotes`.

```
In [ ]: ### GRADED
        ###
        ###

        all_titles_quotes = soup_quotes.find_all('title')


        ### For verifying answer:
        print('all_titles_quotes: {}\n'.format(all_titles_quotes))
        print('type(all_titles_quotes): {}\n'.format(type(all_titles_quotes)))
        for title in all_titles_quotes:
            print(title.text)
```

Section 1.0.1

Question 5 Examining the HTML source of `soup_quotes`, it looks something like this:

```
<table width="100%" border="1" cellpadding="6" cellspacing="0" class="body">
<tr bgcolor="#FFFFFF">
<td>Newsreader [John Cleese]: And now for something completely
different.<br/>
<b>Monty Python's Flying Circus</b></td>
</tr>
<tr>
<td>Norman [Eric Idle]: Is your wife a..."goer"... eh?
Know what I mean? Know what I mean? Nudge nudge. Nudge nudge!
Know what I mean? Say no more...Know what I mean?<br/>
<b>Monty Python's Flying Circus</b></td>
</tr>
<tr>
<td>Norman [Eric Idle]: A nod's as good as a wink to a blind bat,
eh?.<br/>
<b>Monty Python's Flying Circus</b></td>
</tr>

...
</tr>
</table>
```

The quotations referred to in this page, then, are enclosed within a `<table>` tag with `class="body"` (as opposed to other tables contained in the same page that contain, e.g., links to advertisements or pages with more quotations). The quotations are enclosed between `<td>` tags.

You can extract this table using the `find` method (matching on `'table'` and using the keyword argument `class_="body"` to match on the required attribute). From that result, extract the corresponding quotations using the `find_all` method matching on `'td'`. The `find_all` yields a `ResultSet` with the desired quotations as the `text` attributes for all the tags within. + Assign the output of the `find` method to `table_quotes`. + Assign the output of the `find_all` method to `quotes`. + Assign the number of elements in `quotes` to `num_quotes`. + Assign the fifth element (i.e., element 4 when indexed from 0) to `quotes_4`.

```
In [ ]: ### GRADED
        ### Extract the 'table' tag from soup_quotes using the find method with the keyword ar
        ###     assign the result to table_quotes.
        ### Extract all the tags matching "td" from table_quotes using the find_all method;
        ###     assign the result to quotes.
        ### Assign num_quotes and quotes_4 using the length of quotes and the text attribute o
        ###     of quotes respectively.
        ###

        table_quotes = soup_quotes.find('table', class_='body')
        quotes = table_quotes.find_all('td')
        num_quotes = len(quotes)
        quotes_4 = quotes[4].text


        ### For verifying answer:
        print('num_quotes: {}\n'.format(num_quotes))
        print('quotes_4:\n\n{}'.format(quotes_4))
```

Section 1.0.1

Question 6 You will now use `find_all` to extract all the annotations (i.e., with tag `<a>`) and links from `soup_quotes`. * Assign the result of the `find_all` method to `links`. * Assign the number of elements within `links` to the integer `num_links`. * Construct a list `local_links` by extracting the `href` attributes from all the links in `links` that begin with the character `/`. * Assign the number of elements within `local_links` to the integer `num_local`. * Assign the last entry of `local_links` to the identifier `local_last`.

```
In [ ]: ### GRADED
        ### Assign the result of the `find_all` method to `links`.
        ### Assign the number of elements within `links` to the integer `num_links`.
        ### Construct a list `local_links` by extracting the `href` attributes from all the li
        ### Assign the number of elements within `local_links` to the integer `num_local`.
        ### Assign the last entry of `local_links` to the identifier `local_last`.
        ###
```

```
links = soup_quotes.find_all('a')
num_links = len(links)
local_links = [link for link in links if link.get('href').startswith('/')]
num_local = len(local_links)
local_last = local_links[-1].get('href')


### For verifying answer:
print('num_links: {}\n'.format(num_links))
print('num_local: {}\n'.format(num_local))
print('local_last: {}'.format(local_last))
```

Section 1.0.1

1.1.1 Scraping Wikipedia

You'll move on here to scrape a table from Wikipedia.

Question 7 The identifier `URL_solar_system` provides a link to the page on gravitationally rounded objects in the solar system.

To start, download the text of the page into a `BeautifulSoup` object; bind the resulting `str` object to the identifier `soup_solar`.

```
In [ ]: ### GRADED
        ### Use the text attribute of the BeautifulSoup class to extract the body of the
        ###   required web page (provided in URL_solar_system).
        ###
        # Get the soup object first....
        URL_solar_system = "https://en.wikipedia.org/wiki/List_of_gravitationally_rounded_obje

        soup_solar = BeautifulSoup(requests.get(URL_solar_system).text)
```

Section 1.0.1

Question 8 Having constructed the `BeautifulSoup` object `soup_solar`, your task in this question is to extract all the tables. You can use the `find_all` method to get `<table>` tags with the attribute `'class': 'wikitable'`. * Assign result of this top level `find_all` to the identifier `tables`. * Assign the third (i.e., item 2 indexed from 0) to `planets`. * Find all the `<tr>` tags within `planets` and bind result to `rows`.

```
In [ ]: ### GRADED
        ### Assign result of this top level `find_all` to the identifier `tables`.
        ###      Assign the third (i.e., item 2 indexed from 0) to `planets`.
        ###      Find all the `<tr>` tags within `planets` and bind result to `rows`.
        ###

        tables = soup_solar.find_all("table", attrs={'class':'wikitable'})
        planets = tables[2]
        rows = planets.find_all('tr')


        ### For verifying answer:
        print('Number of tables: {}\n'.format(len(tables)))
        print('Number of rows in planets: {}'.format(len(rows)))
```

Section 1.0.1

Question 9 Having extracted the desired sequence of tables from the `BeautifulSoup` object `soup_solar`, now you can extract data from a table. You have already extracted the rows of this table in `rows` in Question 8.

The first row, `rows[0]` contains the headers of the table (which is the names of the planets in the solar system). The column headers in `rows[0]` look something like this when printed:

```
*Mercury[6]

*Venus[7]

*Earth[8]

*Mars[9]

řJupiter[10]

řSaturn[11]

Uranus[12]

Neptune[13]
```

The raw Unicode is given as follows:

```
\n\xa0\n\n*Mercury[6]\n\n*Venus[7]\n\n*Earth[8]\n\n*Mars[9]\n\nřJupiter[10]\n\nřSaturn[11]\n\nU
```

Your task in this question is to extract the column headers (the names of the planets) from `rows[0]`. Each column header is enclosed in a `<th>` tag. In each case, the associated `text` attribute has extraneous characters to ignore (notably the leading Unicode character and the trailing link number in square brackets).

```
In [ ]: ### GRADED
        ### Construct a list headers of strings extracted from soup_solar. The strings, when
        ###     obtained correctly, are the names of the planets in order from nearest to the
        ###     sun. Assign the resulting list of str to the identifier headers.
        ###

        headers = []
        for col in rows[0].find_all('th'):
            label = col.text
            end = label.find('[')
            label = label[1:end]
            if label:
                headers.append(label)


        ### For verifying answer:
        print('headers: {}'.format(headers))
```

Section 1.0.1

Question 10 Skipping to the fourth row (i.e., `rows[3]`), the `BeautifulSoup` elements sought are nested within `td` tags. * Use the `find_all` method to extract all the `<td>` tags from `rows[3]`. * For the eight planet columns, extract the 8 (mean) distances (measured in km) from each celestial body to the sun; store in a list called `mean_distances` sorted as in the Wikipedia article. * For each entry, ignore the second number expressing the distance in astronomical units. * Be sure to transform the `str` data to `float` values.

```
In [ ]: ### GRADED
        ### Extract numbers from 4th row (Mean distance from the Sun)
        ###    Use the find_all method to extract all the td tags from rows[3]
        ###    and accumulate a list mean_distances of each planet obtained by
        ###    converting the first number from each tag to a float.
        ###

        row_3 = rows[3].find_all("td")
        mean_distances = [float(col.text.split()[0].replace(',','')) for col in row_3[2:]]


        ### For verifying answer:
        print('mean_distances: {}'.format(mean_distances))
```

Section 1.0.1

Question 11 Your final task in Question 11 is similar to that in Question 10. You will extract all the entries of `rows[4]` that match `<td>`. * Use the `find_all` method to extract all the `<td>` tags from `rows[4]`; bind the result to `rows_4`. * For the eight planet columns, extract the 8 equatorial radii (measured in km) from each celestial body orbitting the sun; store the result in a list called `equatorial_radii` in the same sequence as in `mean_distances` from Question 10. * For each entry, ignore the second number expressing the distance in astronomical units. * Be sure to transform the `str` data to `float` values.

```
In [ ]: ### GRADED
        ### Extract numbers from 5th row (Equatorial radius).
        ###    Use the find_all method to extract all the td tags from rows[4]
        ###    and accumulate a list equatorial_radii of each planet obtained by
        ###    converting the first number from each tag to a float.

        ###

        row_4 = rows[4].find_all("td")
        equatorial_radii = [float(col.text.split()[0].replace(',','')) for col in row_4[2:]]


        ### For verifying answer:
        print('equatorial_radii: {}'.format(equatorial_radii))
```

Section 1.0.1
