5 Web Scraping I: Introduction to BeautifulSoup

Lab Objective: *Web Scraping is the process of gathering data from websites on the internet. Since almost everything rendered by an internet browser as a web page uses HTML, the first step in web scraping is being able to extract information from HTML. In this lab, we introduce BeautifulSoup, Python's canonical tool for efficiently and cleanly navigating and parsing HTML.*

HTML

Hyper Text Markup Language, or *HTML*, is the standard *markup language*—a language designed for the processing, definition, and presentation of text—for creating webpages. It provides a document with structure and is composed of pairs of *tags* to surround and define various types of content. Opening tags have a tag name surrounded by angle brackets (`<tag-name>`). The companion closing tag looks the same, but with a forward slash before the tag name (`</tag-name>`). A list of all current HTML tags can be found at `http://htmldog.com/reference/htmltags`.

Most tags can be combined with *attributes* to include more data about the content, help identify individual tags, and make navigating the document much simpler. In the following example, the `<a>` tag has `id` and `href` attributes.

```
<html>                                  <!-- Opening tags -->
   <body>
       <p>
           Click <a id='info' href='http://www.example.com'>here</a>
           for more information.
       </p>                             <!-- Closing tags -->
   </body>
</html>
```

In HTML, `href` stands for *hypertext reference*, a link to another website. Thus the above example would be rendered by a browser as a single line of text, with `here` being a clickable link to `http://www.example.com`:

Click here for more information.

Unlike Python, HTML does not enforce indentation (or any whitespace rules), though indentation generally makes HTML more readable. The previous example can even be written equivalently in a single line.

```
<html><body><p>Click <a id='info' href='http://www.example.com/info'>here</a>
  for more information.</p></body></html>
```

Special tags, which don't contain any text or other tags, are written without a closing tag and in a single pair of brackets. A forward slash is included between the name and the closing bracket. Examples of these include `<hr/>`, which describes a horizontal line, and `<img/>`, the tag for representing an image.

Problem 1. The HTML of a website is easy to view in most browsers. In Google Chrome, go to `http://www.example.com`, right click anywhere on the page that isn't a picture or a link, and select `View Page Source`. This will open the HTML source code that defines the page. Examine the source code. What tags are used? What is the value of the `type` attribute associated with the `style` tag?

Write a function that returns the set of names of tags used in the website, and the value of the `type` attribute of the `style` tag (as a string).
(Hint: there are ten unique tag names.)

BeautifulSoup

BeautifulSoup (`bs4`) is a package[1] that makes it simple to navigate and extract data from HTML documents. See `http://www.crummy.com/software/BeautifulSoup/bs4/doc/index.html` for the full documentation.

The `bs4.BeautifulSoup` class accepts two parameters to its constructor: a string of HTML code, and an HTML parser to use under the hood. The HTML parser is technically a keyword argument, but the constructor prints a warning if one is not specified. The standard choice for the parser is `"html.parser"`, which means the object uses the standard library's `html.parser` module as the engine behind the scenes.

Note

Depending on project demands, a parser other than `"html.parser"` may be useful. A couple of other options are `"lxml"`, an extremely fast parser written in C, and `"html5lib"`, a slower parser that treats HTML in much the same way a web browser does, allowing for irregularities. Both must be installed independently; see `https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser` for more information.

A `BeautifulSoup` object represents an HTML document as a tree. In the tree, each tag is a *node* with nested tags and strings as its *children*. The `prettify()` method returns a string that can be printed to represent the BeautifulSoup object in a readable format that reflects the tree structure.

[1] BeautifulSoup is not part of the standard library; install it with `conda install beautifulsoup4` or with `pip install beautifulsoup4`.

```
>>> from bs4 import BeautifulSoup

>>> small_example_html = """
<html><body><p>
    Click <a id='info' href='http://www.example.com'>here</a>
    for more information.
</p></body></html>
"""

>>> small_soup = BeautifulSoup(small_example_html, 'html.parser')
>>> print(small_soup.prettify())
<html>
 <body>
  <p>
   Click
   <a href="http://www.example.com" id="info">
    here
   </a>
   for more information.
  </p>
 </body>
</html>
```

Each tag in a `BeautifulSoup` object's HTML code is stored as a `bs4.element.Tag` object, with actual text stored as a `bs4.element.NavigableString` object. Tags are accessible directly through the `BeautifulSoup` object.

```
# Get the <p> tag (and everything inside of it).
>>> small_soup.p
<p>
    Click <a href="http://www.example.com" id="info">here</a>
    for more information.
</p>

# Get the <a> sub-tag of the <p> tag.
>>> a_tag = small_soup.p.a
>>> print(a_tag, type(a_tag), sep='\n')
<a href="http://www.example.com" id="info">here</a>
<class 'bs4.element.Tag'>

# Get just the name, attributes, and text of the <a> tag.
>>> print(a_tag.name, a_tag.attrs, a_tag.string, sep="\n")
a
{'id': 'info', 'href': 'http://www.example.com'}
here
```

Attribute	Description
`name`	The name of the tag
`attrs`	A dictionary of the attributes
`string`	The single string contained in the tag
`strings`	Generator for strings of children tags
`stripped_strings`	Generator for strings of children tags, stripping whitespace
`text`	Concatenation of strings from all children tags

Table 5.1: Data attributes of the `bs4.element.Tag` class.

Problem 2. The `BeautifulSoup` class has a `find_all()` method that, when called with `True` as the only argument, returns a list of all tags in the HTML source code.

Write a function that accepts a string of HTML code as an argument. Use BeautifulSoup to return a list of the **names** of the tags in the code. Use your function and the source code from `http://www.example.com` (see `example.html`) to check your answers from Problem 1.

Navigating the Tree Structure

Not all tags are easily accessible from a `BeautifulSoup` object. Consider the following example.

```
>>> pig_html = """
<html><head><title>Three Little Pigs</title></head>
<body>
<p class="title"><b>The Three Little Pigs</b></p>
<p class="story">Once upon a time, there were three little pigs named
<a href="http://example.com/larry" class="pig" id="link1">Larry,</a>
<a href="http://example.com/mo" class="pig" id="link2">Mo</a>, and
<a href="http://example.com/curly" class="pig" id="link3">Curly.</a>
<p>The three pigs had an odd fascination with experimental construction.</p>
<p>...</p>
</body></html>
"""

>>> pig_soup = BeautifulSoup(pig_html, "html.parser")
>>> pig_soup.p
<p class="title"><b>The Three Little Pigs</b></p>

>>> pig_soup.a
<a class="pig" href="http://example.com/larry" id="link1">Larry,</a>
```

Since the HTML in this example has several `<p>` and `<a>` tags, only the **first** tag of each name is accessible directly from `pig_soup`. The other tags can be accessed by manually navigating through the HTML tree.

Every HTML tag (except for the topmost tag, which is usually `<html>`) has a *parent* tag. Each tag also has and zero or more *sibling* and *children* tags or text. Following a true tree structure, every `bs4.element.Tag` in a soup has multiple attributes for accessing or iterating through parent, sibling, or child tags.

Attribute	Description
`parent`	The parent tag
`parents`	Generator for the parent tags up to the top level
`next_sibling`	The tag immediately after to the current tag
`next_siblings`	Generator for sibling tags after the current tag
`previous_sibling`	The tag immediately before to the current tag
`previous_siblings`	Generator for sibling tags before the current tag
`contents`	A list of the immediate children tags
`children`	Generator for immediate children tags
`descendants`	Generator for all children tags (recursively)

Table 5.2: Navigation attributes of the `bs4.element.Tag` class.

```
>>> print(pig_soup.prettify())
<html>
 <head>                     # <head> is the parent of the <title>
  <title>
   Three Little Pigs
  </title>
 </head>
 <body>                     # <body> is the sibling of <head>
  <p class="title">         # and the parent of two <p> tags (title and story).
   <b>
    The Three Little Pigs
   </b>
  </p>
  <p class="story">
   Once upon a time, there were three little pigs named
   <a class="pig" href="http://example.com/larry" id="link1">
    Larry,
   </a>
   <a class="pig" href="http://example.com/mo" id="link2">
    Mo
   </a>
   , and
   <a class="pig" href="http://example.com/curly" id="link3">
    Curly.                  # The preceding <a> tags are siblings with each
   </a>                     # other and the following two <p> tags.
   <p>
    The three pigs had an odd fascination with experimental construction.
   </p>
   <p>
    ...
   </p>
  </p>
 </body>
</html>
```

```
# Start at the first <a> tag in the soup.
>>> a_tag = pig_soup.a
>>> a_tag
<a class="pig" href="http://example.com/larry" id="link1">Larry,</a>

# Get the names of all of <a>'s parent tags, traveling up to the top.
# The name '[document]' means it is the top of the HTML code.
>>> [par.name for par in a_tag.parents]     # <a>'s parent is <p>, whose
['p', 'body', 'html', '[document]']         # parent is <body>, and so on.

# Get the next siblings of <a>.
>>> a_tag.next_sibling
'\n'                                        # The first sibling is just text.
>>> a_tag.next_sibling.next_sibling         # The second sibling is a tag.
<a class="pig" href="http://example.com/mo" id="link2">Mo</a>

# Alternatively, get all siblings past <a> at once.
>>> list(a_tag.next_siblings)
['\n',
 <a class="pig" href="http://example.com/mo" id="link2">Mo</a>,
 ', and\n',
 <a class="pig" href="http://example.com/curly" id="link3">Curly.</a>,
 '\n',
 <p>The three pigs had an odd fascination with experimental construction.</p>,
 '\n',
 <p>...</p>,
 '\n']
```

Note carefully that newline characters are considered to be children of a parent tag. Therefore iterating through children or siblings often requires checking which entries are tags and which are just text.

```
# Get to the <p> tag that has class="story".
>>> p_tag = pig_soup.body.p.next_sibling.next_sibling
>>> p_tag.attrs["class"]                # Make sure it's the right tag.
['story']

# Iterate through the child tags of <p> and print hrefs whenever they exist.
>>> for child in p_tag.children:
...     if hasattr(child, "attrs") and "href" in child.attrs:
...         print(child.attrs["href"])
http://example.com/larry
http://example.com/mo
http://example.com/curly
```

Note that the `"class"` attribute of the `<p>` tag is a list. This is because the `"class"` attribute can take on several values at once; for example, the tag `<p class="story book">` is of class `'story'` and of class `'book'`.

Note

The behavior of the `string` attribute of a `bs4.element.Tag` object depends on the structure of the corresponding HTML tag.

1. If the tag has a string of text and no other child elements, then `string` is just that text.
2. If the tag has exactly one child tag and the child tag has only a string of text, then the tag has the same `string` as its child tag.
3. If the tag has more than one child, then `string` is `None`. In this case, use `strings` to iterate through the child strings. Alternatively, the `get_text()` method returns all text belonging to a tag and to all of its descendants. In other words, it returns anything inside a tag that isn't another tag.

```
>>> pig_soup.head
<head><title>Three Little Pigs</title></head>

# Case 1: the <title> tag's only child is a string.
>>> pig_soup.head.title.string
'Three Little Pigs'

# Case 2: The <head> tag's only child is the <title> tag.
>>> pig_soup.head.string
'Three Little Pigs'

# Case 3: the <body> tag has several children.
>>> pig_soup.body.string is None
True
>>> print(pig_soup.body.get_text().strip())
The Three Little Pigs
Once upon a time, there were three little pigs named
Larry,
Mo, and
Curly.
The three pigs had an odd fascination with experimental construction.
...
```

Problem 3. The file `example.html` contains the HTML source for `http://www.example.com`. Write a function that reads the file and loads the code into BeautifulSoup. Find the only `<a>` tag with a hyperlink and return its text.

Searching for Tags

Navigating the HTML tree manually can be helpful for gathering data out of lists or tables, but these kinds of structures are usually buried deep in the tree. The `find()` and `find_all()` methods of the `BeautifulSoup` class identify tags that have distinctive characteristics, making it much easier to jump straight to a desired location in the HTML code. The `find()` method only returns the **first** tag that matches a given criteria, while `find_all()` returns a list of all matching tags. Tags can be matched by name, attributes, and/or text.

```
# Find the first <b> tag in the soup.
>>> pig_soup.find(name='b')
<b>The Three Little Pigs</b>

# Find all tags with a class attribute of 'pig'.
# Since 'class' is a Python keyword, use 'class_' as the argument.
>>> pig_soup.find_all(class_="pig")
[<a class="pig" href="http://example.com/larry" id="link1">Larry,</a>,
 <a class="pig" href="http://example.com/mo" id="link2">Mo</a>,
 <a class="pig" href="http://example.com/curly" id="link3">Curly.</a>]

# Find the first tag that matches several attributes.
>>> pig_soup.find(attrs={"class": "pig", "href": "http://example.com/mo"})
<a class="pig" href="http://example.com/mo" id="link2">Mo</a>

# Find the first tag whose text is 'Mo'.
>>> pig_soup.find(string='Mo')
'Mo'                                  # The result is the actual string,
>>> soup.find(string='Mo').parent       # so go up one level to get the tag.
<a class="pig" href="http://example.com/mo" id="link2">Mo</a>
```

Problem 4. The file `san_diego_weather.html` contains the HTML source for an old page from Weather Underground.[a]. Write a function that reads the file and loads it into BeautifulSoup. Return a list of the following tags:

1. The tag containing the date "Thursday, January 1, 2015".
2. The tags which contain the **links** "Previous Day" and "Next Day."
3. The tag which contains the number associated with the Actual Max Temperature.

This HTML tree is significantly larger than the previous examples. To get started, consider opening the file in a web browser. Find the element that you are searching for on the page, right click it, and select `Inspect`. This opens the HTML source at the element that the mouse clicked on.

[a] See `http://www.wunderground.com/history/airport/KSAN/2015/1/1/DailyHistory.html?req_city=San+Diego&req_state=CA&req_statename=California&reqdb.zip=92101&reqdb.magic=1&reqdb.wmo=99999&MR=1`

Advanced Search Techniques

Consider the problem of finding the tag that is a link the URL `http://example.com/curly`.

```
>>> pig_soup.find(href="http://example.com/curly")
<a class="pig" href="http://example.com/curly" id="link3">Curly.</a>
```

This approach works, but it requires entering in the entire URL. To perform generalized searches, the `find()` and `find_all()` method also accept compile regular expressions from the `re` module. This way, the methods locate tags whose name, attributes, and/or string matches a pattern.

```
>>> import re

# Find the first tag with an href attribute containing 'curly'.
>>> pig_soup.find(href=re.compile(r"curly"))
<a class="pig" href="http://example.com/curly" id="link3">Curly.</a

# Find the first tag with a string that starts with 'Cu'.
>>> pig_soup.find(string=re.compile(r"^Cu")).parent
<a class="pig" href="http://example.com/curly" id="link3">Curly.</a>

# Find all tags with text containing 'Three'.
>>> [tag.parent for tag in pig_soup.find_all(string=re.compile(r"Three"))]
[<title>Three Little Pigs</title>, <b>The Three Little Pigs</b>]
```

Finally, to find a tag that has a particular attribute, regardless of the actual value of the attribute, use `True` in place of search values.

```
# Find all tags with an 'id' attribute.
>>> pig_soup.find_all(id=True)
[<a class="pig" href="http://example.com/larry" id="link1">Larry,</a>,
 <a class="pig" href="http://example.com/mo" id="link2">Mo</a>,
 <a class="pig" href="http://example.com/curly" id="link3">Curly.</a>]

# Final the names all tags WITHOUT an 'id' attribute.
>>> [tag.name for tag in pig_soup.find_all(id=False)]
['html', 'head', 'title', 'body', 'p', 'b', 'p', 'p', 'p']
```

Problem 5. The file `large_banks_index.html` is an index of data about large banks, as recorded by the Federal Reserve.[a] Write a function that reads the file and loads the source into BeautifulSoup. Return a list of the tags containing the links to bank data from September 30, 2003 to December 31, 2014, where the dates are in reverse chronological order.

[a] See `https://www.federalreserve.gov/releases/lbr/`.

Problem 6. The file `large_banks_data.html` is one of the pages from the index in Problem 5.[a] Write a function that reads the file and loads the source into BeautifulSoup. Create a single figure with two subplots:

1. A sorted bar chart of the seven banks with the most domestic branches.
2. A sorted bar chart of the seven banks with the most foreign branches.

In the case of a tie, sort the banks alphabetically by name.

[a] See `http://www.federalreserve.gov/releases/lbr/20030930/default.htm`.