Web Scraping Lab

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be:

- Familiar with the basics of the `BeautifulSoup` Python library
- Be able to scrape webpages for data and filter the data

For this lab, we are going to be using Python and several Python libraries. Some of these libraries might be installed in your lab environment or in SN Labs. Others may need to be installed by you. The cells below will install these libraries when executed.

```
!pip install bs4
#!pip install requests
```

```
Requirement already satisfied: bs4 in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (0.0.2)
Requirement already satisfied: beautifulsoup4 in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from
bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from
beautifulsoup4->bs4) (2.3.2.post1)
```

Import the required modules and functions

```
from bs4 import BeautifulSoup # this module helps in web scrapping.
import requests  # this module helps us to download a web page
```

Beautiful Soup is a Python library for pulling data out of HTML and XML files, we will focus on HTML files. This is accomplished by representing the HTML as a set of objects with methods used to parse the HTML. We can navigate the HTML as a tree, and/or filter out what we are looking for.

Consider the following HTML:

```
%%html
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'>Lebron James</b></h3>
```

```
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

```
<IPython.core.display.HTML object>
```

We can store it as a string in the variable HTML:

```
html="<!DOCTYPE html><html><head><title>Page
Title</title></head><body><h3><b id='boldest'>Lebron James</b></h3><p>
Salary: $ 92,000,000 </p><h3> Stephen Curry</h3><p> Salary: $85,000,
000 </p><h3> Kevin Durant </h3><p> Salary: $73,200,
000</p></body></html>"
```

To parse a document, pass it into the BeautifulSoup constructor. The BeautifulSoup object represents the document as a nested data structure:

```
soup = BeautifulSoup(html, 'html.parser')
```

First, the document is converted to Unicode (similar to ASCII) and HTML entities are converted to Unicode characters. Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. The BeautifulSoup object can create other types of objects. In this lab, we will cover BeautifulSoup and Tag objects, that for the purposes of this lab are identical. Finally, we will look at NavigableString objects.

We can use the method prettify() to display the HTML in the nested structure:

```
print(soup.prettify())
```

```
<!DOCTYPE html>
<html>
 <head>
  <title>
   Page Title
  </title>
 </head>
 <body>
  <h3>
   <b id="boldest">
    Lebron James
   </b>
  </h3>
  <p>
   Salary: $ 92,000,000
  </p>
```

```
  <h3>
   Stephen Curry
  </h3>
  <p>
   Salary: $85,000, 000
  </p>
  <h3>
   Kevin Durant
  </h3>
  <p>
   Salary: $73,200, 000
  </p>
 </body>
</html>
```

Tags

Let's say we want the title of the page and the name of the top paid player. We can use the Tag. The Tag object corresponds to an HTML tag in the original document, for example, the tag title.

```
tag_object=soup.title
print("tag object:",tag_object)
```

```
tag object: <title>Page Title</title>
```

we can see the tag type bs4.element.Tag

```
print("tag object type:",type(tag_object))
```

```
tag object type: <class 'bs4.element.Tag'>
```

If there is more than one Tag with the same name, the first element with that Tag name is called. This corresponds to the most paid player:

```
tag_object=soup.h3
tag_object
```

```
<h3><b id="boldest">Lebron James</b></h3>
```

Enclosed in the bold attribute b, it helps to use the tree representation. We can navigate down the tree using the child attribute to get the name.

Children, Parents, and Siblings

As stated above, the Tag object is a tree of objects. We can access the child of the tag or navigate down the branch as follows:

```
tag_child =tag_object.b
tag_child

<b id="boldest">Lebron James</b>
```

You can access the parent with the parent

```
parent_tag=tag_child.parent
parent_tag

<h3><b id="boldest">Lebron James</b></h3>
```

this is identical to:

```
tag_object

<h3><b id="boldest">Lebron James</b></h3>
```

tag_object parent is the body element.

```
tag_object.parent

<body><h3><b id="boldest">Lebron James</b></h3><p> Salary: $
92,000,000 </p><h3> Stephen Curry</h3><p> Salary: $85,000, 000
</p><h3> Kevin Durant </h3><p> Salary: $73,200, 000</p></body>
```

tag_object sibling is the paragraph element

```
sibling_1=tag_object.next_sibling
sibling_1

<p> Salary: $ 92,000,000 </p>
```

`sibling_2` is the `header` element, which is also a sibling of both `sibling_1` and `tag_object`

```
sibling_2=sibling_1.next_sibling
sibling_2

<h3> Stephen Curry</h3>
```

Use the object sibling_2 and the method next_sibling to find the salary of Stephen Curry:

```
sibling_2.next_sibling

<p> Salary: $85,000, 000 </p>
```

HTML Attributes

If the tag has attributes, the tag id="boldest" has an attribute id whose value is boldest. You can access a tag's attributes by treating the tag like a dictionary:

```
tag_child['id']
```

```
'boldest'
```

You can access that dictionary directly as attrs:

```
tag_child.attrs
```

```
{'id': 'boldest'}
```

You can also work with Multi-valued attributes. Check out [1] for more.

We can also obtain the content of the attribute of the tag using the Python get() method.

```
tag_child.get('id')
```

```
'boldest'
```

Navigable String

A string corresponds to a bit of text or content within a tag. Beautiful Soup uses the NavigableString class to contain this text. In our HTML we can obtain the name of the first player by extracting the string of the Tag object tag_child as follows:

```
tag_string=tag_child.string
tag_string
```

```
'Lebron James'
```

we can verify the type is Navigable String

```
type(tag_string)
```

```
bs4.element.NavigableString
```

A NavigableString is similar to a Python string or Unicode string. To be more precise, the main difference is that it also supports some BeautifulSoup features. We can convert it to string object in Python:

```
unicode_string = str(tag_string)
unicode_string
```

```
'Lebron James'
```

Filters allow you to find complex patterns, the simplest filter is a string. In this section we will pass a string to a different filter method and Beautiful Soup will perform a match against that exact string. Consider the following HTML of rocket launches:

```
%%html
<table>
  <tr>
    <td id='flight' >Flight No</td>
    <td>Launch site</td>
    <td>Payload mass</td>
   </tr>
  <tr>
    <td>1</td>
    <td><a
href='https://en.wikipedia.org/wiki/Florida'>Florida</a></td>
    <td>300 kg</td>
  </tr>
  <tr>
    <td>2</td>
    <td><a href='https://en.wikipedia.org/wiki/Texas'>Texas</a></td>
    <td>94 kg</td>
  </tr>
  <tr>
    <td>3</td>
    <td><a href='https://en.wikipedia.org/wiki/Florida'>Florida<a>
</td>
    <td>80 kg</td>
  </tr>
</table>
```

```
<IPython.core.display.HTML object>
```

We can store it as a string in the variable table:

```
table="<table><tr><td id='flight'>Flight No</td><td>Launch site</td>
<td>Payload mass</td></tr><tr> <td>1</td><td><a
href='https://en.wikipedia.org/wiki/Florida'>Florida<a></td><td>300
kg</td></tr><tr><td>2</td><td><a
href='https://en.wikipedia.org/wiki/Texas'>Texas</a></td><td>94
kg</td></tr><tr><td>3</td><td><a
href='https://en.wikipedia.org/wiki/Florida'>Florida<a> </td><td>80
kg</td></tr></table>"

table_bs = BeautifulSoup(table, 'html.parser')
```

find All

The find_all() method looks through a tag's descendants and retrieves all descendants that match your filters.

Name

When we set the name parameter to a tag name, the method will extract all the tags with that name and its children.

```
table_rows=table_bs.find_all('tr')
table_rows
```

```
[<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload
mass</td></tr>,
 <tr> <td>1</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td><t
d>300 kg</td></tr>,
 <tr><td>2</td><td><a
href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94
kg</td></tr>,
 <tr><td>3</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>80 kg</td></tr>]
```

The result is a Python Iterable just like a list, each element is a tag object:

```
first_row =table_rows[0]
first_row
```

```
<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload
mass</td></tr>
```

The type is tag

```
print(type(first_row))
```

```
<class 'bs4.element.Tag'>
```

we can obtain the child

```
first_row.td
```

```
<td id="flight">Flight No</td>
```

If we iterate through the list, each element corresponds to a row in the table:

```
for i,row in enumerate(table_rows):
    print("row",i,"is",row)
```

```
row 0 is <tr><td id="flight">Flight No</td><td>Launch site</td>
<td>Payload mass</td></tr>
row 1 is <tr> <td>1</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td><t
```

```
d>300 kg</td></tr>
row 2 is <tr><td>2</td><td><a
href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94
kg</td></tr>
row 3 is <tr><td>3</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>80 kg</td></tr>
```

As row is a cell object, we can apply the method find_all to it and extract table cells in the object cells using the tag td, this is all the children with the name td. The result is a list, each element corresponds to a cell and is a Tag object, we can iterate through this list as well. We can extract the content using the string attribute.

```
for i,row in enumerate(table_rows):
    print("row",i)
    cells=row.find_all('td')
    for j,cell in enumerate(cells):
        print('colunm',j,"cell",cell)
```

```
row 0
colunm 0 cell <td id="flight">Flight No</td>
colunm 1 cell <td>Launch site</td>
colunm 2 cell <td>Payload mass</td>
row 1
colunm 0 cell <td>1</td>
colunm 1 cell <td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td>
colunm 2 cell <td>300 kg</td>
row 2
colunm 0 cell <td>2</td>
colunm 1 cell <td><a
href="https://en.wikipedia.org/wiki/Texas">Texas</a></td>
colunm 2 cell <td>94 kg</td>
row 3
colunm 0 cell <td>3</td>
colunm 1 cell <td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a></td>
colunm 2 cell <td>80 kg</td>
```

If we use a list we can match against any item in that list.

```
list_input=table_bs .find_all(name=["tr", "td"])
list_input
```

```
[<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload
mass</td></tr>,
 <td id="flight">Flight No</td>,
 <td>Launch site</td>,
 <td>Payload mass</td>,
```

```
 <tr> <td>1</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td><t
d>300 kg</td></tr>,
 <td>1</td>,
 <td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td>,
 <td>300 kg</td>,
 <tr><td>2</td><td><a
href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94
kg</td></tr>,
 <td>2</td>,
 <td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td>,
 <td>94 kg</td>,
 <tr><td>3</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>80 kg</td></tr>,
 <td>3</td>,
 <td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td>,
 <td>80 kg</td>]
```

Attributes

If the argument is not recognized it will be turned into a filter on the tag's attributes. For example with the id argument, Beautiful Soup will filter against each tag's id attribute. For example, the first td elements have a value of id of flight, therefore we can filter based on that id value.

```
table_bs.find_all(id="flight")
```

```
[<td id="flight">Flight No</td>]
```

We can find all the elements that have links to the Florida Wikipedia page:

```
list_input=table_bs.find_all(href="https://en.wikipedia.org/wiki/
Florida")
list_input
```

```
[<a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a>,
 <a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a>]
```

If we set the href attribute to True, regardless of what the value is, the code finds all tags with href value:

```
table_bs.find_all(href=True)
```

```
[<a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a>,
 <a href="https://en.wikipedia.org/wiki/Texas">Texas</a>,
 <a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a>]
```

There are other methods for dealing with attributes and other related methods. Check out the following link

Using the logic above, find all the elements without href value

```
table_bs.find_all(href=False)
```

```
[<table><tr><td id="flight">Flight No</td><td>Launch site</td>
<td>Payload mass</td></tr><tr> <td>1</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td><t
d>300 kg</td></tr><tr><td>2</td><td><a
href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94
kg</td></tr><tr><td>3</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>80 kg</td></tr></table>,
 <tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload
mass</td></tr>,
 <td id="flight">Flight No</td>,
 <td>Launch site</td>,
 <td>Payload mass</td>,
 <tr> <td>1</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td><t
d>300 kg</td></tr>,
 <td>1</td>,
 <td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></td>,
 <a></a>,
 <td>300 kg</td>,
 <tr><td>2</td><td><a
href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94
kg</td></tr>,
 <td>2</td>,
 <td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td>,
 <td>94 kg</td>,
 <tr><td>3</td><td><a
href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>80 kg</td></tr>,
 <td>3</td>,
 <td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td>,
 <a> </a>,
 <td>80 kg</td>]
```

Using the soup object soup, find the element with the id attribute content set to "boldest".

```
soup.find_all(id="boldest")
```

```
[<b id="boldest">Lebron James</b>]
```

string

With string you can search for strings instead of tags, where we find all the elments with Florida:

```
table_bs.find_all(string="Florida")
```

```
['Florida', 'Florida']
```

find

The find_all() method scans the entire document looking for results. It's useful if you are looking for one element, as you can use the find() method to find the first element in the document. Consider the following two tables:

```
%%html
<h3>Rocket Launch </h3>

<p>
<table class='rocket'>
  <tr>
    <td>Flight No</td>
    <td>Launch site</td>
    <td>Payload mass</td>
  </tr>
  <tr>
    <td>1</td>
    <td>Florida</td>
    <td>300 kg</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Texas</td>
    <td>94 kg</td>
  </tr>
  <tr>
    <td>3</td>
    <td>Florida </td>
    <td>80 kg</td>
  </tr>
</table>
</p>
<p>

<h3>Pizza Party  </h3>
```

```
<table class='pizza'>
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
   </tr>
  <tr>
    <td>Domino's Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td >144 </td>
  </tr>
  <tr>
    <td>Papa John's </td>
    <td>15 </td>
    <td>165</td>
  </tr>
```

```
<IPython.core.display.HTML object>
```

We store the HTML as a Python string and assign two_tables:

```
two_tables="<h3>Rocket Launch </h3><p><table
class='rocket'><tr><td>Flight No</td><td>Launch site</td> <td>Payload
mass</td></tr><tr><td>1</td><td>Florida</td><td>300
kg</td></tr><tr><td>2</td><td>Texas</td><td>94
kg</td></tr><tr><td>3</td><td>Florida </td><td>80
kg</td></tr></table></p><p><h3>Pizza Party  </h3><table
class='pizza'><tr><td>Pizza Place</td><td>Orders</td> <td>Slices
</td></tr><tr><td>Domino's
Pizza</td><td>10</td><td>100</td></tr><tr><td>Little
Caesars</td><td>12</td><td >144 </td></tr><tr><td>Papa John's
</td><td>15 </td><td>165</td></tr>"
```

We create a BeautifulSoup object two_tables_bs

```
two_tables_bs= BeautifulSoup(two_tables, 'html.parser')
```

We can find the first table using the tag name table

```
two_tables_bs.find("table")
```

```
<table class="rocket"><tr><td>Flight No</td><td>Launch site</td>
<td>Payload mass</td></tr><tr><td>1</td><td>Florida</td><td>300
```

```
kg</td></tr><tr><td>2</td><td>Texas</td><td>94
kg</td></tr><tr><td>3</td><td>Florida </td><td>80 kg</td></tr></table>
```

We can filter on the class attribute to find the second table, but because class is a keyword in Python, we add an underscore to differentiate them.

```
two_tables_bs.find("table",class_='pizza')
```

```
<table class="pizza"><tr><td>Pizza Place</td><td>Orders</td>
<td>Slices </td></tr><tr><td>Domino's
Pizza</td><td>10</td><td>100</td></tr><tr><td>Little
Caesars</td><td>12</td><td>144 </td></tr><tr><td>Papa John's
</td><td>15 </td><td>165</td></tr></table>
```

We Download the contents of the web page:

```
url = "http://www.ibm.com"
```

We use get to download the contents of the webpage in text format and store in a variable called data:

```
data  = requests.get(url).text
```

We create a BeautifulSoup object using the BeautifulSoup constructor

```
soup = BeautifulSoup(data,"html5lib")  # create a soup object using
the variable 'data'
```

```
---------------------------------------------------------------------------
-----
FeatureNotFound                             Traceback (most recent call
last)
/tmp/ipykernel_1072/1112302729.py in <module>
----> 1 soup = BeautifulSoup(data,"html5lib")  # create a soup object
using the variable 'data'

~/conda/envs/python/lib/python3.7/site-packages/bs4/__init__.py in
__init__(self, markup, features, builder, parse_only, from_encoding,
exclude_encodings, element_classes, **kwargs)
    249                    "Couldn't find a tree builder with the
features you "
    250                    "requested: %s. Do you need to install a
parser library?"
--> 251                    % ",".join(features))
    252
    253         # At this point either we have a TreeBuilder instance
in
```

```
FeatureNotFound: Couldn't find a tree builder with the features you
requested: html5lib. Do you need to install a parser library?
```

Scrape all links

```
for link in soup.find_all('a',href=True):  # in html anchor/link is
represented by the tag <a>

    print(link.get('href'))
```

Scrape all images Tags

```
for link in soup.find_all('img'):# in html image is represented by the
tag <img>
    print(link)
    print(link.get('src'))
```

Scrape data from HTML tables

```
#The below url contains an html table with data about colors and color
codes.
url = "https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/datasets/
HTMLColorCodes.html"
```

Before proceeding to scrape a web site, you need to examine the contents and the way data is organized on the website. Open the above url in your browser and check how many rows and columns there are in the color table.

```
# get the contents of the webpage in text format and store in a
variable called data
data  = requests.get(url).text

soup = BeautifulSoup(data,"html5lib")

#find a html table in the web page
table = soup.find('table') # in html table is represented by the tag
<table>

#Get all rows from the table
for row in table.find_all('tr'): # in html table row is represented by
the tag <tr>
    # Get all columns in each row.
    cols = row.find_all('td') # in html a column is represented by the
tag <td>
    color_name = cols[2].string # store the value in column 3 as
color_name
    color_code = cols[3].string # store the value in column 4 as
```

```
color_code
    print("{}--->{}".format(color_name,color_code))
```

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-17	0.1	Joseph Santarcangelo	Created initial version of the lab