



Web Scraping Lab

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

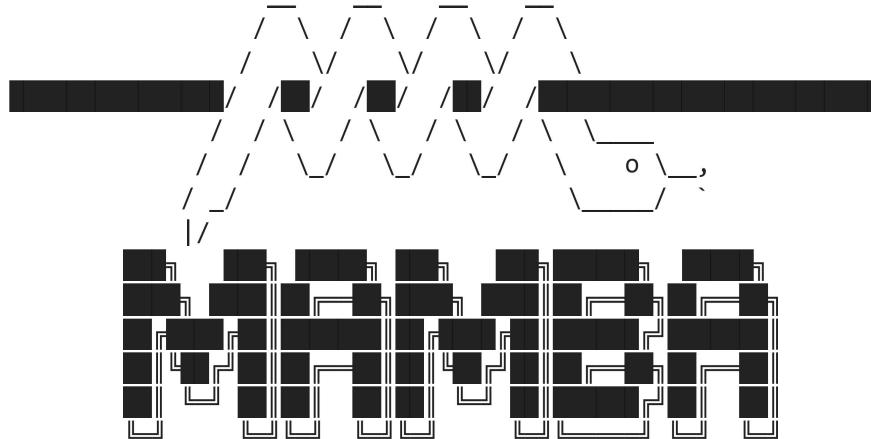
Table of Contents

- Beautiful Soup Object
 - Tag
 - Children, Parents, and Siblings
 - HTML Attributes
 - Navigable String
- Filter
 - findAll
 - find
 - HTML Attributes
 - Navigable String
- Downloading And Scraping The Contents Of A Web

Estimated time needed: **25 min**

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.

```
In [1]: !mamba install bs4==4.10.0 -y  
!pip install lxml==4.6.4  
!mamba install html5lib==1.1 -y  
# !pip install requests==2.26.0
```



mamba (0.15.3) supported by @QuantStack

GitHub: <https://github.com/mamba-org/mamba>

Twitter: <https://twitter.com/QuantStack>

Looking for: ['bs4==4.10.0']

```
pkgs/main/linux-64      [=>          ] (00m:00s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [<=>        ] (00m:00s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/noarch        [<=>        ] (00m:00s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/noarch        [=>          ] (00m:00s) 704 KB / ?? (2.17 MB/s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/noarch        [=>          ] (00m:00s) 704 KB / ?? (2.17 MB/s)
pkgs/r/linux-64          [<=>        ] (00m:00s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/noarch        [=>          ] (00m:00s) 704 KB / ?? (2.17 MB/s)
pkgs/r/linux-64          [=>          ] (00m:00s) 668 KB / ?? (2.05 MB/s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/noarch        [<=>        ] (00m:00s) Finalizing...
pkgs/r/linux-64          [=>          ] (00m:00s) 668 KB / ?? (2.05 MB/s)
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
pkgs/main/noarch        [<=>        ] (00m:00s) Done
pkgs/r/linux-64          [=>          ] (00m:00s) 668 KB / ?? (2.05 MB/s)
pkgs/main/noarch        [=====] (00m:00s) Done
pkgs/main/linux-64      [=>          ] (00m:00s) 580 KB / ?? (1.88 MB/s)
pkgs/r/noarch           [=>          ] (00m:00s) 608 KB / ?? (1.89 MB/s)
```

Pinned packages:

Transaction

Prefix: /home/jupyterlab/conda/envs/pythom

Updating specs:

- bs4==4.10.0
- ca-certificates
- certifi
- openssl

Package	Version	Build	Channel	Size
<hr/>				
Install:				
<hr/>				
+ bs4	4.10.0	hd3eb1b0_0	pkgs/main/noarch	10 KB
<hr/>				
Upgrade:				
<hr/>				
- ca-certificates	2022.9.24	ha878542_0	installed	
+ ca-certificates	2023.01.10	h06a4308_0	pkgs/main/linux-64	120 KB
- certifi	2022.9.24	pyhd8ed1ab_0	installed	
+ certifi	2022.12.7	py37h06a4308_0	pkgs/main/linux-64	150 KB
- openssl	1.1.1s	h0b41bf4_1	installed	
+ openssl	1.1.1t	h7f8727e_0	pkgs/main/linux-64	4 MB
<hr/>				
Downgrade:				
<hr/>				
- beautifulsoup4	4.11.1	pyha770c72_0	installed	
+ beautifulsoup4	4.10.0	pyh06a4308_0	pkgs/main/noarch	85 KB
<hr/>				
Summary:				
<hr/>				
Install: 1 packages				
Upgrade: 3 packages				
Downgrade: 1 packages				
<hr/>				
Total download: 4 MB				
<hr/>				

```

Downloading  [>                                         ] (00m:00s) 48.81 KB/s
Extracting  [>                                         ] (---)
Finished bs4                                         (00m:00s) 10 KB    49
KB/s
Downloading  [>                                         ] (00m:00s) 48.81 KB/s
Extracting  [>                                         ] (---)
Downloading  [>                                         ] (00m:00s) 48.81 KB/s
Extracting  [>                                         ] (---)
Downloading  [>                                         ] (00m:00s) 48.81 KB/s
Extracting  [>                                         ] (---)
Downloading  [>                                         ] (00m:00s) 441.74 KB/s
Extracting  [>                                         ] (---)
Finished beautifulsoup4                           (00m:00s) 85 KB    395
KB/s
Downloading  [>                                         ] (00m:00s) 441.74 KB/s
Extracting  [>                                         ] (---)
Downloading  [>                                         ] (00m:00s) 441.74 KB/s

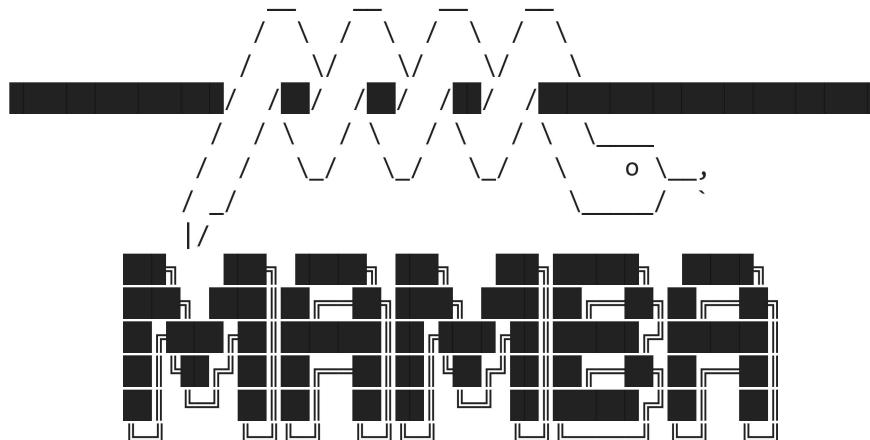
```

```

Extracting    [>                               ] (--)          ] (---)
Downloading  [==>                            ] (00m:00s)  974.62 KB/s
Extracting    [>                               ] (--)          ] (---)
Downloading  [==>                            ] (00m:00s)  974.62 KB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Finished certifi (00m:00s)                   150 KB     636
KB/s
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Finished ca-certificates (00m:00s)           120 KB     545
KB/s
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   1 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   2 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   2 / 5
Downloading  [====>                          ] (00m:00s)  1.51 MB/s
Extracting    [=====>                         ] (00m:00s)   2 / 5
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   2 / 5
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   3 / 5
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   4 / 5
Finished openssl (00m:00s)                   4 MB      13
MB/s
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   4 / 5
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   4 / 5
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   4 / 5
Downloading  [=====>                         ] (00m:00s)  13.83 MB/s
Extracting    [=====>                         ] (00m:00s)   5 / 5
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting lxml==4.6.4
    Downloading lxml-4.6.4-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.man
ylinux_2_24_x86_64.whl (6.3 MB)
    6.3/6.3 MB 31.7 MB/s eta 0:00:0000:0
100:01
Installing collected packages: lxml
    Attempting uninstall: lxml
        Found existing installation: lxml 4.9.1
        Uninstalling lxml-4.9.1:
            Successfully uninstalled lxml-4.9.1

```

```
Successfully installed lxml-4.6.4
```



mamba (0.15.3) supported by @QuantStack

GitHub: <https://github.com/mamba-org/mamba>
Twitter: <https://twitter.com/QuantStack>

Looking for: ['html5lib==1.1']

```
pkgs/main/noarch      [>          ] (---) No change
pkgs/main/noarch      [=====] (00m:00s) No change
pkgs/r/linux-64       [>          ] (---) No change
pkgs/r/linux-64       [=====] (00m:00s) No change
pkgs/main/linux-64   [>          ] (---) No change
pkgs/main/linux-64   [=====] (00m:00s) No change
pkgs/r/noarch         [>          ] (---) No change
pkgs/r/noarch         [=====] (00m:00s) No change
```

Pinned packages:

- python 3.7.*

Transaction

Prefix: /home/jupyterlab/conda/envs/python

Updating specs:

- html5lib==1.1
- ca-certificates
- certifi
- openssl

Package	Version	Build	Channel	Size
<hr/>				
Install:				

```
+ html5lib      1.1    pyhd3eb1b0_0   pkgs/main/noarch      91 KB
+ webencodings  0.5.1   py37_1        pkgs/main/linux-64    19 KB
```

Summary:

Install: 2 packages

Total download: 110 KB

```
Downloading [=====] (00m:00s) 454.11 KB/s
Extracting [>
Finished html5lib          (00m:00s)      91 KB    455
KB/s
Downloading [=====] (00m:00s) 454.11 KB/s
Extracting [>
Downloading [=====] (00m:00s) 454.11 KB/s
Extracting [>
Downloading [=====] (00m:00s) 454.11 KB/s
Extracting [>
Downloading [=====] (00m:00s) 523.41 KB/s
Extracting [>
Finished webencodings       (00m:00s)      19 KB    91
KB/s
Downloading [=====] (00m:00s) 523.41 KB/s
Extracting [>
Downloading [=====] (00m:00s) 523.41 KB/s
Extracting [>
Downloading [=====] (00m:00s) 523.41 KB/s
Extracting [>           ] (00m:00s)      1 / 2
Downloading [=====] (00m:00s) 523.41 KB/s
Extracting [>           ] (00m:00s)      1 / 2
Downloading [=====] (00m:00s) 523.41 KB/s
Extracting [>           ] (00m:00s)      2 / 2
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Import the required modules and functions

```
In [5]: from bs4 import BeautifulSoup # this module helps in web scrapping.
import requests # this module helps us to download a web page
```

Beautiful Soup Objects

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:

```
In [6]: %%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>
```

Lebron James

Salary: \$ 92,000,000

Stephen Curry

Salary: \$85,000, 000

Kevin Durant

Salary: \$73,200, 000

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

```
In [7]: html="<!DOCTYPE html><html><head><title>Page Title</title></head><body><h3><b id='b
```

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

```
In [8]: 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, and `NavigableString` objects.

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

```
In [9]: 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`.

```
In [10]: tag_object=soup.title
print("tag object:",tag_object)
```

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

we can see the tag type `bs4.element.Tag`

```
In [11]: 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:

```
In [12]: tag_object=soup.h3
tag_object
```

```
Out[12]: <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:

```
In [13]: tag_child =tag_object.b
tag_child
```

```
Out[13]: <b id="boldest">Lebron James</b>
```

You can access the parent with the `parent`

```
In [14]: parent_tag=tag_child.parent
parent_tag
```

```
Out[14]: <h3><b id="boldest">Lebron James</b></h3>
```

this is identical to

```
In [15]: tag_object
```

```
Out[15]: <h3><b id="boldest">Lebron James</b></h3>
```

`tag_object` parent is the `body` element.

```
In [16]: tag_object.parent
```

```
Out[16]: <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

```
In [17]: sibling_1=tag_object.next_sibling
sibling_1
```

```
Out[17]: <p> Salary: $ 92,000,000 </p>
```

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

```
In [18]: sibling_2=sibling_1.next_sibling  
sibling_2
```

```
Out[18]: <h3> Stephen Curry</h3>
```

Exercise: `next_sibling`

Using the object `sibling_2` and the property `next_sibling` to find the salary of Stephen Curry:

```
In [ ]:
```

► Click here for the solution

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:

```
In [19]: tag_child['id']
```

```
Out[19]: 'boldest'
```

You can access that dictionary directly as `attrs`:

```
In [20]: tag_child.attrs
```

```
Out[20]: {'id': 'boldest'}
```

You can also work with Multi-valued attribute check out [\[1\]](#) for more.

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

```
In [21]: tag_child.get('id')
```

```
Out[21]: 'boldest'
```

Navigable String

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

```
In [22]: tag_string=tag_child.string
tag_string
```

```
Out[22]: 'Lebron James'
```

we can verify the type is Navigable String

```
In [23]: type(tag_string)
```

```
Out[23]: bs4.element.NavigableString
```

A NavigableString is just like 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:

```
In [24]: unicode_string = str(tag_string)
unicode_string
```

```
Out[24]: 'Lebron James'
```

Filter

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 BeautifulSoup will perform a match against that exact string. Consider the following HTML of rocket launches:

```
In [25]: %%html


|                                                                                                                                         |                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Launch site</td> <td>Payload mass&lt;/td&gt; </td>                                                                                      | Payload mass</td>                                                                                                       |
| 1</td> <td><a &gt;florida&lt;="" <td="" a&gt;&lt;="" href="https://en.wikipedia.org/wiki/Florida" td&gt;="">300 kg&lt;/td&gt; </a></td> | <a &gt;florida&lt;="" <td="" a&gt;&lt;="" href="https://en.wikipedia.org/wiki/Florida" td&gt;="">300 kg&lt;/td&gt; </a> |
| 2</td> <td><a &gt;texas&lt;="" <td="" a&gt;&lt;="" href="https://en.wikipedia.org/wiki/Texas" td&gt;="">94 kg&lt;/td&gt; </a></td>      | <a &gt;texas&lt;="" <td="" a&gt;&lt;="" href="https://en.wikipedia.org/wiki/Texas" td&gt;="">94 kg&lt;/td&gt; </a>      |
| 3</td> <td><a &gt;florida&lt;="" <td="" a&gt;&lt;="" href="https://en.wikipedia.org/wiki/Florida" td&gt;="">80 kg&lt;/td&gt; </a></td>  | <a &gt;florida&lt;="" <td="" a&gt;&lt;="" href="https://en.wikipedia.org/wiki/Florida" td&gt;="">80 kg&lt;/td&gt; </a>  |


```

Flight No	Launch site	Payload mass
1	Florida	300 kg
2	Texas	94 kg
3	Florida	80 kg

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

```
In [26]: table=<table><tr><td id='flight'>Flight No</td><td>Launch site</td> <td>Payload ma
```

```
In [27]: 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.

The Method signature for `find_all(name, attrs, recursive, string, limit, **kwargs)`

Name

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

```
In [28]: table_rows=table_bs.find_all('tr')
table_rows
```

```
Out[28]: [<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>
</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> </a>
</a></td><td>80 kg</td></tr>]
```

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

```
In [29]: first_row =table_rows[0]
first_row
```

```
Out[29]: <tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>
```

The type is `tag`

```
In [30]: print(type(first_row))
<class 'bs4.element.Tag'>
```

we can obtain the child

```
In [31]: first_row.td
```

```
Out[31]: <td id="flight">Flight No</td>
```

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

```
In [32]: 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></td><td>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></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.

```
In [33]: for i, row in enumerate(table_rows):
    print("row", i)
    cells = row.find_all('td')
    for j, cell in enumerate(cells):
        print('column', j, "cell", cell)
```

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

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

```
In [34]: list_input=table_bs .find_all(name=["tr", "td"])
list_input
```

```
Out[34]: [<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><td>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 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.

```
In [35]: table_bs.find_all(id="flight")
```

```
Out[35]: [<td id="flight">Flight No</td>]
```

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

```
In [36]: list_input=table_bs.find_all(href="https://en.wikipedia.org/wiki/Florida")
list_input
```

```
Out[36]: [<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:

```
In [37]: table_bs.find_all(href=True)
```

```
Out[37]: [<a href="https://en.wikipedia.org/wiki/Florida">Florida</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](#)

Exercise: find_all

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

► Click here for the solution

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

► Click here for the solution

string

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

```
In [38]: table_bs.find_all(string="Florida")
```

```
Out[38]: ['Florida', 'Florida']
```

find

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

```
In [39]: %%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>






```

Rocket Launch

Flight No	Launch site	Payload mass
1	Florida	300 kg
2	Texas	94 kg
3	Florida	80 kg

Pizza Party

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

```
In [40]: 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></table>
```

We create a `BeautifulSoup` object `two_tables_bs`

```
In [41]: two_tables_bs= BeautifulSoup(two_tables, 'html.parser')
```

We can find the first table using the tag name `table`

```
In [42]: 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.

```
In [43]: 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>
```

Downloading And Scraping The Contents Of A Web Page

We Download the contents of the web page:

```
In [44]: 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`:

```
In [45]: data = requests.get(url).text
```

We create a `BeautifulSoup` object using the `BeautifulSoup` constructor

```
In [46]: soup = BeautifulSoup(data,"html.parser") # create a soup object using the variable 'data'
```

Scrape all links

```
In [47]: for link in soup.find_all('a',href=True): # in html anchor/link is represented by the tag <a>
    print(link.get('href'))
```

```
https://www.ibm.com/products/turbonomic
https://www.ibm.com/products/api-connect
https://www.ibm.com/resources/automation-innovation-workshop
https://www.ibm.com/automation
https://www.ibm.com/downloads/cas/RZQKRMKO
#
#
#
#
#
https://www.ibm.com/employment/?lnk=flatitem
https://www.ibm.com/impact
https://research.ibm.com/
https://www.ibm.com/consulting/?lnk=flath1
https://www.ibm.com/consulting/strategy/?lnk=flath1
https://www.ibm.com/consulting/ibmix?lnk=flath1
https://www.ibm.com/consulting/technology/
https://www.ibm.com/consulting/operations/?lnk=flath1
https://www.ibm.com/strategic-partnerships
https://www.ibm.com/products/instana
https://www.ibm.com/products/turbonomic
https://www.ibm.com/automation
https://www.ibm.com/products/envizi
https://www.ibm.com/products/maximo
https://www.ibm.com/sustainability
https://www.ibm.com/cloud/campaign/cloud-simplicity
https://www.ibm.com/cloud/campaign/cloud-simplicity
http://ibm.com/consulting/ibmix
https://www.ibm.com/consulting
https://www.ibm.com/sports/masters
https://www.ibm.com/cloud/watson-knowledge-catalog
https://www.ibm.com/data-management
https://www.ibm.com/data-fabric
https://www.ibm.com/
```

Scrape all images Tags

```
In [48]: for link in soup.find_all('img'):# in html image is represented by
the tag <img>
    print(link)
    print(link.get('src'))
```

```

https://1.dam.s81c.com/p/0b526ddf40c413f5/security-leadspace-desktop.jpg.global.mr_5x4.jpg

https://1.dam.s81c.com/p/0b52088cc24c2f3d/CS-The-Automate-Beer-Question-30_ProRes_frame-13-22.png.global.sr_5x4.png

https://1.dam.s81c.com/p/0b526ddf40c4140c/CS_The_Sustainability_Building_Question_30_ProRes-frame-07-00.png.global.sr_5x4.png

https://1.dam.s81c.com/p/0b52088cc24c2f41/CS_The_Modernize_Banking_Question_V1_30_ProRes_frame-00-06.png.global.sr_5x4.png

https://1.dam.s81c.com/p/0b52088cc24c2f37/CS_the_transform_masters_question_30_prores_frame00.png.global.mr_5x4.jpg

https://1.dam.s81c.com/p/0b52088cc24c2f35/CS_The_Modernize_Banking_Question_V1_30_ProRes-frame-04-30.png.global.sr_5x4.png

https://1.dam.s81c.com/p/0aac9cf57bcbf324/dotcom-1-overview.jpg.global.sr_16x9.jpg

https://1.dam.s81c.com/p/0aac9cf57c4bf381/0935e0d55846b9ec_dotcom-2-strategy.jpg.global.m_16x9.jpg

https://1.dam.s81c.com/p/0aac9cf57c4bf382/0656655b47a3af8b_dotcom-3-experience.jpg.global.sr_16x9.jpg

https://1.dam.s81c.com/p/0aac9cf57c4bf383/08f9513598a70746_dotcom-4-technology.jpg.global.sr_16x9.jpg

https://1.dam.s81c.com/p/0aac9cf57c4bf380/0935e0d55846b9e4_dotcom-5-operations.jpg.global.sr_16x9.jpg

https://1.dam.s81c.com/p/0aac9cf57c4bf381/0935e0d55846b9e5_dotcom-6-communication.jpg.global.sr_16x9.jpg
```

```
y="bx--image-12" class="bx--image__img" src="https://1.dam.s81c.com/p/0aac9cf5  
7c4bf384/064e0139f5a3aa3f_dotcom-6-partners.jpg.global.sr_16x9.jpg"/>  
https://1.dam.s81c.com/p/0aac9cf57c4bf384/064e0139f5a3aa3f_dotcom-6-partners.j  
pg.global.sr_16x9.jpg
```

Scrape data from HTML tables

```
In [49]: #The below url contains an html table with data about colors and co  
lor codes.  
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomai  
n.cloud/IBM-DA0321EN-SkillsNetwork/labs/datasets/HTMLColorCodes.htm  
l"
```

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 are there in the color table.

```
In [50]: # get the contents of the webpage in text format and store in a var  
iable called data  
data = requests.get(url).text
```

```
In [51]: soup = BeautifulSoup(data,"html.parser")
```

```
In [52]: #find a html table in the web page  
table = soup.find('table') # in html table is represented by the ta  
g <table>
```

```
In [53]: #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 co  
lor_name  
    color_code = cols[3].string # store the value in column 4 as co  
lor_code  
    print("{}--->{}".format(color_name,color_code))
```

```

Color Name--->None
lightsalmon--->#FFA07A
salmon--->#FA8072
darksalmon--->#E9967A
lightcoral--->#F08080
coral--->#FF7F50
tomato--->#FF6347
orangered--->#FF4500
gold--->#FFD700
orange--->#FFA500
darkorange--->#FF8C00
lightyellow--->#FFFFE0
lemonchiffon--->#FFFACD
papayawhip--->#FFEFDD
moccasin--->#FFE4B5
peachpuff--->#FFDAB9
palegoldenrod--->#EEE8AA
khaki--->#F0E68C
darkkhaki--->#BDB76B
yellow--->#FFFF00
lawngreen--->#7CFC00
chartreuse--->#7FFF00
limegreen--->#32CD32
lime--->#00FF00
forestgreen--->#228B22
green--->#008000
powderblue--->#B0E0E6
lightblue--->#ADD8E6
lightskyblue--->#87CEFA
skyblue--->#87CEEB
deepskyblue--->#00BFFF
lightsteelblue--->#B0C4DE
dodgerblue--->#1E90FF

```

Scrape data from HTML tables into a DataFrame using BeautifulSoup and Pandas

```
In [54]: import pandas as pd
```

```
In [55]: #The below url contains html tables with data about world population.
url = "https://en.wikipedia.org/wiki/World_population"
```

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 the tables on the webpage.

```
In [56]: # get the contents of the webpage in text format and store in a variable called data
data = requests.get(url).text
```

```
In [57]: soup = BeautifulSoup(data,"html.parser")
```

```
In [58]: #find all html tables in the web page
tables = soup.find_all('table') # in html table is represented by the tag <table>
```

```
In [59]: # we can see how many tables were found by checking the length of the tables list
len(tables)
```

24

Assume that we are looking for the `10 most densely populated countries` table, we can look through the tables list and find the right one we are looking for based on the data in each table or we can search for the table name if it is in the table but this option might not always work.

```
In [60]: for index,table in enumerate(tables):
    if ("10 most densely populated countries" in str(table)):
        table_index = index
print(table_index)
```

4

See if you can locate the table name of the table, `10 most densely populated countries`, below.

```
In [61]: print(tables[table_index].prettify())
```

```
<table class="wikitable sortable" style="text-align:right">
<caption>
  10 most densely populated countries
  <small>
    (with population above 5 million)
  </small>
  <sup class="reference" id="cite_ref-:10_107-0">
    <a href="#cite_note-:10-107">
      [102]
    </a>
  </sup>
</caption>
<tbody>
<tr>
  <th scope="col">
    Rank
  </th>
  <th scope="col">
    Country
  </th>
  <th scope="col">
    Population
  </th>
  <th scope="col">
    Area
    <br/>
    <small>
      (km
      <sup>
        2
      </sup>
      )
    </small>
  </th>
  <th scope="col">
    Density
    <br/>
    <small>
      (pop/km
      <sup>
        2
      </sup>
      )
    </small>
  </th>
</tr>
<tr>
  <td>
    1
  </td>
  <td align="left">
    <span class="flagicon">
      
    </span>
    <a href="/wiki/Singapore" title="Singapore">
        Singapore
    </a>
</td>
<td>
    5,921,231
</td>
<td>
    719
</td>
<td>
    8,235
</td>
</tr>
<tr>
<td>
    2
</td>
<td align="left">
    <span class="flagicon">
        
    </span>
    <a href="/wiki/Bangladesh" title="Bangladesh">
        Bangladesh
    </a>
</td>
<td>
    165,650,475
</td>
<td>
    148,460
</td>
<td>
    1,116
</td>
</tr>
<tr>
<td>
    3
</td>
<td align="left">
    <p>
        <span class="flagicon">
            
    </span>
    <a href="/wiki/State_of_Palestine" title="State of Palestine">
        Palestine
    </a>
    <sup class="reference" id="cite_ref-108">
        <a href="#cite_note-108">
            [103]
        </a>
    </sup>
</p>
</td>
<td>
    5,223,000
</td>
<td>
    6,025
</td>
<td>
    867
</td>
</tr>
<tr>
    <td>
        4
    </td>
    <td align="left">
        <span class="flagicon">
            
        </span>
        <a href="/wiki/Taiwan" title="Taiwan">
            Taiwan
        </a>
    </td>
    <td>
        23,580,712
    </td>
    <td>
        35,980
    </td>
    <td>
        655
    </td>
</tr>
<tr>
    <td>
        5
    </td>
    <td align="left">
```

```
<span class="flagicon">
    
</span>
<a href="/wiki/South_Korea" title="South Korea">
    South Korea
</a>
</td>
<td>
    51,844,834
</td>
<td>
    99,720
</td>
<td>
    520
</td>
</tr>
<tr>
    <td>
        6
    </td>
    <td align="left">
        <span class="flagicon">
            
        </span>
        <a href="/wiki/Lebanon" title="Lebanon">
            Lebanon
        </a>
    </td>
    <td>
        5,296,814
    </td>
    <td>
        10,400
    </td>
    <td>
        509
    </td>
</tr>
<tr>
    <td>
        7
    </td>
    <td align="left">
        <span class="flagicon">
```

```

</span>
<a href="/wiki/Rwanda" title="Rwanda">
    Rwanda
</a>
</td>
<td>
    13,173,730
</td>
<td>
    26,338
</td>
<td>
    500
</td>
</tr>
<tr>
<td>
    8
</td>
<td align="left">
    <span class="flagicon">
        
    </span>
    <a href="/wiki/Burundi" title="Burundi">
        Burundi
    </a>
</td>
<td>
    12,696,478
</td>
<td>
    27,830
</td>
<td>
    456
</td>
</tr>
<tr>
<td>
    9
</td>
<td align="left">
    <span class="flagicon">
        
    </span>
    <a href="/wiki/India" title="India">
        India
    </a>
</td>
<td>
    1,389,637,446
</td>
<td>
    3,287,263
</td>
<td>
    423
</td>
</tr>
<tr>
<td>
    10
</td>
<td align="left">
    <span class="flagicon">
        
    </span>
    <a href="/wiki/Netherlands" title="Netherlands">
        Netherlands
    </a>
</td>
<td>
    17,400,824
</td>
<td>
    41,543
</td>
<td>
    419
</td>
</tr>
</tbody>
</table>

```

```

In [62]: population_data = pd.DataFrame(columns=["Rank", "Country", "Population", "Area", "Density"])

for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")

```

```

if (col != []):
    rank = col[0].text
    country = col[1].text
    population = col[2].text.strip()
    area = col[3].text.strip()
    density = col[4].text.strip()
    population_data = population_data.append({"Rank":rank, "Country":country, "Population":population, "Area":area, "Density":density}, ignore_index=True)

population_data

```

	Pizza Place	Orders	Slices
	Domino's Pizza	10	100
	Little Caesars	12	144
	Papa John's	15	165

	Rank	Country	Population	Area	Density
0	1	Singapore	5,921,231	719	8,235
1	2	Bangladesh	165,650,475	148,460	1,116
2	3	\n Palestine[103]\n\n	5,223,000	6,025	867
3	4	Taiwan	23,580,712	35,980	655
4	5	South Korea	51,844,834	99,720	520
5	6	Lebanon	5,296,814	10,400	509
6	7	Rwanda	13,173,730	26,338	500
7	8	Burundi	12,696,478	27,830	456
8	9	India	1,389,637,446	3,287,263	423
9	10	Netherlands	17,400,824	41,543	419

Scrape data from HTML tables into a DataFrame using BeautifulSoup and read_html

Using the same `url`, `data`, `soup`, and `tables` object as in the last section we can use the `read_html` function to create a DataFrame.

Remember the table we need is located in `tables[table_index]`

We can now use the `pandas` function `read_html` and give it the string version of the table as well as the `flavor` which is the parsing engine `bs4`.

```
In [63]: pd.read_html(str(tables[5]), flavor='bs4')
```

```
Out[63]: [   Rank          Country  Population  Area(km2)  Density(pop/km2) \
0    1            India  1389637446  3287263           423
1    2        Pakistan  242923845  796095           305
2    3      Bangladesh  165650475  148460          1116
3    4            Japan  124214766  377915           329
4    5  Philippines  114597229  300000           382
5    6        Vietnam  103808319  331210           313
6    7  United Kingdom  67791400  243610           278
7    8    South Korea  51844834  99720            520
8    9        Taiwan  23580712  35980            655
9   10      Sri Lanka  23187516  65610            353
```

```
Population trend[citation needed]
0                  Growing
1        Rapidly growing
2                  Growing
3      Declining[104]
4                  Growing
5                  Growing
6                  Growing
7                  Steady
8                  Steady
9                  Growing ]
```

The function `read_html` always returns a list of DataFrames so we must pick the one we want out of the list.

```
In [64]: population_data_read_html = pd.read_html(str(tables[5]), flavor='bs4')[0]

population_data_read_html
```

	Rank	Country	Population	Area(km2)	Density(pop/km2)	Population trend[citation needed]
0	1	India	1389637446	3287263	423	Growing
1	2	Pakistan	242923845	796095	305	Rapidly growing
2	3	Bangladesh	165650475	148460	1116	Growing
3	4	Japan	124214766	377915	329	Declining[104]
4	5	Philippines	114597229	300000	382	Growing
5	6	Vietnam	103808319	331210	313	Growing
6	7	United Kingdom	67791400	243610	278	Growing
7	8	South Korea	51844834	99720	520	Steady
8	9	Taiwan	23580712	35980	655	Steady
9	10	Sri Lanka	23187516	65610	353	Growing

Scrape data from HTML tables into a DataFrame using `read_html`

We can also use the `read_html` function to directly get DataFrames from a `url`.

```
In [65]: dataframe_list = pd.read_html(url, flavor='bs4')
```

We can see there are 25 DataFrames just like when we used `find_all` on the `soup` object.

```
In [66]: len(dataframe_list)
```

```
Out[66]: 24
```

Finally we can pick the DataFrame we need out of the list.

```
In [67]: dataframe_list[5]
```

	Rank	Country	Population	Area(km2)	Density(pop/km2)	Population trend[citation needed]
0	1	India	1389637446	3287263	423	Growing
1	2	Pakistan	242923845	796095	305	Rapidly growing
2	3	Bangladesh	165650475	148460	1116	Growing
3	4	Japan	124214766	377915	329	Declining[104]
4	5	Philippines	114597229	300000	382	Growing
5	6	Vietnam	103808319	331210	313	Growing
6	7	United Kingdom	67791400	243610	278	Growing
7	8	South Korea	51844834	99720	520	Steady
8	9	Taiwan	23580712	35980	655	Steady
9	10	Sri Lanka	23187516	65610	353	Growing

We can also use the `match` parameter to select the specific table we want. If the table contains a string matching the text it will be read.

```
In [68]: pd.read_html(url, match="10 most densely populated countries", flavor='bs4')[0]
```

Out[68]:

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5921231	719	8235
1	2	Bangladesh	165650475	148460	1116
2	3	Palestine[103]	5223000	6025	867
3	4	Taiwan	23580712	35980	655
4	5	South Korea	51844834	99720	520
5	6	Lebanon	5296814	10400	509
6	7	Rwanda	13173730	26338	500
7	8	Burundi	12696478	27830	456
8	9	India	1389637446	3287263	423
9	10	Netherlands	17400824	41543	419

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-08-04	0.2		Made changes to markdown of nextsibling
2020-10-17	0.1	Joseph Santarcangelo	Created initial version of the lab

Copyright © 2020 IBM Corporation. This notebook and its source code are released under the terms of the [MIT License](#).

In []:

In []: