WebScraping_Review_Lab

September 1, 2025

1 Web Scraping Lab

Estimated time needed: **30** minutes

1.1 Objectives

After completing this lab you will be able to:

Table of Contents

```
<ul>
    <li>
        <a href="https://bso/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=0000
        <ul>
            <li>Tag</li>
            <li>Children, Parents, and Siblings</li>
            <li>HTML Attributes</li>
            <li>Navigable String</li>
        </ul>
    </li>
 </ul>
<ul>
    <li>
        <a href="https://filter/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=00
        <ul>
            <li>find All</li>
            <li>find </li>
            <li>HTML Attributes</li>
            <li>Navigable String</li>
        </ul>
    </li>
 </ul>
 <ul>
    <li>
        <a href="https://dscw/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=0000
</li>
     </ul>
<p>
    Estimated time needed: <strong>25 min</strong>
```

</p>

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.

[1]:

```
!mamba install bs4==4.10.0 -y
!pip install lxml==4.6.4
!mamba install html5lib==1.1 -y
!pip install pandas
# !pip install requests==2.26.0
```

```
[+] 0.0s
[+] 0.1s
conda-forge/linux-64                 14.1kB /  46.3MB @ 251.0kB/s  0.0s
[+] 0.2s         0.0 B /  ??.?MB @  ??.?MB/s  0.0s
conda-forge/linux-64                  5.3MB /  46.3MB @  33.0MB/s  0.1s
[+] 0.3s         3.3MB /  21.9MB @  20.9MB/s  0.1s
conda-forge/linux-64                  9.1MB /  46.3MB @  34.2MB/s  0.2s
[+] 0.4s         7.9MB /  21.9MB @  29.6MB/s  0.2s
conda-forge/linux-64                 13.5MB /  46.3MB @  36.7MB/s  0.3s
[+] 0.5s        12.3MB /  21.9MB @  33.4MB/s  0.3s
conda-forge/linux-64                 18.1MB /  46.3MB @  38.0MB/s  0.4s
[+] 0.6s        16.6MB /  21.9MB @  35.3MB/s  0.4s
conda-forge/linux-64                 22.5MB /  46.3MB @  38.8MB/s  0.6s
conda-forge/noarch
21.9MB @  36.6MB/s  0.6s
[+] 0.7s
[+] 0.8s            23.4MB /  46.3MB @  35.3MB/s  0.6s
[+] 0.9s            29.9MB /  46.3MB @  39.2MB/s  0.7s
[+] 1.0s            36.8MB /  46.3MB @  42.5MB/s  0.8s
[+] 1.1s            45.4MB /  46.3MB @  46.9MB/s  0.9s
[+] 1.2s            45.4MB /  46.3MB @  46.9MB/s  1.0s
[+] 1.3s            45.4MB /  46.3MB @  46.9MB/s  1.1s
[+] 1.4s            45.4MB /  46.3MB @  46.9MB/s  1.2s
[+] 1.5s            45.4MB /  46.3MB @  46.9MB/s  1.3s
[+] 1.6s            45.4MB /  46.3MB @  46.9MB/s  1.4s
[+] 1.7s            45.4MB /  46.3MB @  46.9MB/s  1.5s
[+] 1.8s            45.4MB /  46.3MB @  46.9MB/s  1.6s
[+] 1.9s            45.4MB /  46.3MB @  46.9MB/s  1.7s
[+] 2.0s            45.4MB /  46.3MB @  46.9MB/s  1.8s
[+] 2.1s            45.4MB /  46.3MB @  46.9MB/s  1.9s
[+] 2.2s            45.4MB /  46.3MB @  46.9MB/s  2.0s
[+] 2.3s            45.4MB /  46.3MB @  46.9MB/s  2.1s
[+] 2.4s            45.4MB /  46.3MB @  46.9MB/s  2.2s
[+] 2.5s            45.4MB /  46.3MB @  46.9MB/s  2.3s
[+] 2.6s            45.4MB /  46.3MB @  46.9MB/s  2.4s
[+] 2.7s            45.4MB /  46.3MB @  46.9MB/s  2.5s
[+] 2.8s            45.4MB /  46.3MB @  46.9MB/s  2.6s
[+] 2.9s             45.4MB /  46.3MB @  46.9MB/s  2.7s
[+] 3.0s             45.4MB /  46.3MB @  46.9MB/s  2.8s
[+] 3.1s             45.4MB /  46.3MB @  46.9MB/s  2.9s
[+] 3.2s             45.4MB /  46.3MB @  46.9MB/s  3.0s
[+] 3.3s             45.4MB /  46.3MB @  46.9MB/s  3.1s
[+] 3.4s             45.4MB /  46.3MB @  46.9MB/s  3.2s
[+] 3.5s             45.4MB /  46.3MB @  46.9MB/s  3.3s
[+] 3.6s             45.4MB /  46.3MB @  46.9MB/s  3.4s
[+] 3.7s             45.4MB /  46.3MB @  46.9MB/s  3.5s
[+] 3.8s             45.4MB /  46.3MB @  46.9MB/s  3.6s
[+] 3.9s             45.4MB /  46.3MB @  46.9MB/s  3.7s
[+] 4.0s             45.4MB /  46.3MB @  46.9MB/s  3.8s
[+] 4.1s             45.4MB /  46.3MB @  46.9MB/s  3.9s
[+] 4.2s             45.4MB /  46.3MB @  46.9MB/s  4.0s
[+] 4.3s             45.4MB /  46.3MB @  46.9MB/s  4.1s
[+] 4.4s             45.4MB /  46.3MB @  46.9MB/s  4.2s
[+] 4.5s             45.4MB /  46.3MB @  46.9MB/s  4.3s
[+] 4.6s             45.4MB /  46.3MB @  46.9MB/s  4.4s
[+] 4.7s             45.4MB /  46.3MB @  46.9MB/s  4.5s
[+] 4.8s             45.4MB /  46.3MB @  46.9MB/s  4.6s
[+] 4.9s             45.4MB /  46.3MB @  46.9MB/s  4.7s
[+] 5.0s             45.4MB /  46.3MB @  46.9MB/s  4.8s
[+] 5.1s             45.4MB /  46.3MB @  46.9MB/s  4.9s
[+] 5.2s             45.4MB /  46.3MB @  46.9MB/s  5.0s
[+] 5.3s             45.4MB /  46.3MB @  46.9MB/s  5.1s
[+] 5.4s             45.4MB /  46.3MB @  46.9MB/s  5.2s
[+] 5.5s             45.4MB /  46.3MB @  46.9MB/s  5.3s
[+] 5.6s             45.4MB /  46.3MB @  46.9MB/s  5.4s
[+] 5.7s             45.4MB /  46.3MB @  46.9MB/s  5.5s
[+] 5.8s             45.4MB /  46.3MB @  46.9MB/s  5.6s
[+] 5.9s             45.4MB /  46.3MB @  46.9MB/s  5.7s
[+] 6.0s             45.4MB /  46.3MB @  46.9MB/s  5.8s
[+] 6.1s             45.4MB /  46.3MB @  46.9MB/s  5.9s
[+] 6.2s             45.4MB /  46.3MB @  46.9MB/s  6.0s
[+] 6.3s             45.4MB /  46.3MB @  46.9MB/s  6.1s
[+] 6.4s             45.4MB /  46.3MB @  46.9MB/s  6.2s
[+] 6.5s             45.4MB /  46.3MB @  46.9MB/s  6.3s
[+] 6.6s             45.4MB /  46.3MB @  46.9MB/s  6.4s
[+] 6.7s             45.4MB /  46.3MB @  46.9MB/s  6.5s
[+] 6.8s             45.4MB /  46.3MB @  46.9MB/s  6.6s
[+] 6.9s             45.4MB /  46.3MB @  46.9MB/s  6.7s
[+] 7.0s             45.4MB /  46.3MB @  46.9MB/s  6.8s
[+] 7.1s             45.4MB /  46.3MB @  46.9MB/s  6.9s
[+] 7.2s             45.4MB /  46.3MB @  46.9MB/s  7.0s
[+] 7.3s             45.4MB /  46.3MB @  46.9MB/s  7.1s
[+] 7.4s             45.4MB /  46.3MB @  46.9MB/s  7.2s
[+] 7.5s             45.4MB /  46.3MB @  46.9MB/s  7.3s
[+] 7.6s             45.4MB /  46.3MB @  46.9MB/s  7.4s
[+] 7.7s             45.4MB /  46.3MB @  46.9MB/s  7.5s
[+] 7.8s             45.4MB /  46.3MB @  46.9MB/s  7.6s
[+] 7.9s             45.4MB /  46.3MB @  46.9MB/s  7.7s
[+] 8.0s             45.4MB /  46.3MB @  46.9MB/s  7.8s
[+] 8.1s             45.4MB /  46.3MB @  46.9MB/s  7.9s
[+] 8.2s             45.4MB /  46.3MB @  46.9MB/s  8.0s
[+] 8.3s             45.4MB /  46.3MB @  46.9MB/s  8.1s
[+] 8.4s             45.4MB /  46.3MB @  46.9MB/s  8.2s
[+] 8.5s             45.4MB /  46.3MB @  46.9MB/s  8.3s
[+] 8.6s             45.4MB /  46.3MB @  46.9MB/s  8.4s
[+] 8.7s             45.4MB /  46.3MB @  46.9MB/s  8.5s
[+] 8.8s             45.4MB /  46.3MB @  46.9MB/s  8.6s
conda-forge/linux-64                                46.3MB @
46.9MB/s  8.8s
DEPRECATION: --no-python-version-warning is deprecated. pip 25.1 will enforce
this behaviour change. A possible replacement is to remove the flag as it's a
no-op. Discussion can be found at https://github.com/pypa/pip/issues/13154

Pinned packages:

  - python=3.12

Pinned packages:

  - python=3.12


Transaction

  Prefix: /opt/conda

  Updating specs:

   - bs4==4.10.0


  Package           Version  Build         Channel          Size

  Install:


  + bs4              4.10.0  hd8ed1ab_0    conda-forge       4kB

  Downgrade:


  - beautifulsoup4   4.12.3  pyha770c72_1  conda-forge     118kB
  + beautifulsoup4   4.10.0  pyha770c72_0  conda-forge      79kB
  Summary:

  Install: 1 packages
  Downgrade: 1 packages

  Total download: 84kB





Transaction starting
[+] 0.0s
Downloading      -----------------------
0.0 B                           0.0s
beautifulsoup4
79.2kB @ 240.4kB/s  0.0s.0s
bs4                                                     4.3kB @  ??.?MB/s  0.0s
[+] 0.1s
Downloading      -----------------------  83.5kB
0.0s
Unlinking
beautifulsoup4-4.12.3-pyha770c72_1--------      0 beautifulsoup4
0.0s
Linking beautifulsoup4-4.10.0-pyha770c72_0
Linking bs4-4.10.0-hd8ed1ab_0

Transaction finished

Collecting lxml==4.6.4
  Downloading lxml-4.6.4.tar.gz (3.2 MB)
                           3.2/3.2 MB
93.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) … error
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
   exit code: 1
   > [5 lines of output]
      /tmp/pip-
install-d0ollezm/lxml_b7e474d0a6a9482da6ff6262580b4c76/setup.py:67:
DeprecationWarning: pkg_resources is deprecated as an API. See
https://setuptools.pypa.io/en/latest/pkg_resources.html
        import pkg_resources
      Building lxml version 4.6.4.
      Building without Cython.
      Error: Please make sure the libxml2 and libxslt development
```

```
packages are installed.
      [end of output]

  note: This error originates from a subprocess, and is likely not a
problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
 > See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
conda-forge/linux-64                                        Using cache
conda-forge/noarch                                          Using cache
DEPRECATION: --no-python-version-warning is deprecated. pip
25.1 will enforce this behaviour change. A possible replacement is to remove the
flag as it's a no-op. Discussion can be found at
https://github.com/pypa/pip/issues/13154

Pinned packages:

  - python=3.12

Pinned packages:

  - python=3.12


Transaction

  Prefix: /opt/conda

  Updating specs:

   - html5lib==1.1


  Package     Version  Build         Channel         Size

  Install:


  + html5lib      1.1  pyhd8ed1ab_2  conda-forge     95kB

  Summary:

  Install: 1 packages
  Total download: 95kB




Transaction starting
[+] 0.0s
Downloading
-----------------------   0.0 B
0.0s
html5lib
94.9kB @ 167.0kB/s  0.0s
[+] 0.1s
Downloading      -----------------------  94.9kB
0.0s
Linking
html5lib-1.1-pyhd8ed1ab_2000;000;000m-------------------       0 html5lib
0.0s

Transaction finished

Collecting pandas
  Downloading
pandas-2.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(91 kB)
Collecting numpy>=1.26.0 (from pandas)
  Downloading
numpy-2.3.2-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata
(62 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2024.2)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading
pandas-2.3.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.0
MB)
                          12.0/12.0 MB
23.6 MB/s eta 0:00:00:00:01
Downloading
numpy-2.3.2-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (16.6
MB)
                          16.6/16.6 MB
6.6 MB/s eta 0:00:000:00:01
```

```
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, numpy, pandas
Successfully installed numpy-2.3.2 pandas-2.3.2 tzdata-2025.2
```

Import the required modules and functions

[2]:

```
pip install --upgrade beautifulsoup4
```

```
Requirement already satisfied: beautifulsoup4 in /opt/conda/lib/python3.12/site-
packages (4.10.0)
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.13.5-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/lib/python3.12/site-
packages (from beautifulsoup4) (2.5)
Requirement already satisfied: typing-extensions>=4.0.0 in
/opt/conda/lib/python3.12/site-packages (from beautifulsoup4) (4.12.2)
Downloading beautifulsoup4-4.13.5-py3-none-any.whl (105 kB)
Installing collected packages: beautifulsoup4
  Attempting uninstall: beautifulsoup4
    Found existing installation: beautifulsoup4 4.10.0
    Uninstalling beautifulsoup4-4.10.0:
      Successfully uninstalled beautifulsoup4-4.10.0
Successfully installed beautifulsoup4-4.13.5
Note: you may need to restart the kernel to use updated packages.
```

[3]:

```
pip install --upgrade pandas
```

```
Requirement already satisfied: pandas in /opt/conda/lib/python3.12/site-packages
(2.3.2)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2.3.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.12/site-
packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

suppress all warnings

[4]:

```python
import warnings
warnings.simplefilter("ignore")
```

[5]:

```python
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:

[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>
```

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

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

[7]:

```
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, which represents the document as a nested data structure:

[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:

[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>
```

1.2 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.

[10]:

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

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

we can see the tag type bs4.element.Tag

[11]:

```python
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:

[12]:

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

[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.

1.2.1 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:

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

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

You can access the parent with the parent

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

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

this is identical to

[15]:
```
tag_object
```

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

tag_object parent is the body element.

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

[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

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

[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`

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

[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:

```
[19]: sibling_2.next_sibling
```

```
[19]: <p> Salary: $85,000, 000 </p>
```

Click here for the solution

```
sibling_2.next_sibling
```

1.2.2 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:

```
[20]: tag_child['id']
```

```
[20]: 'boldest'
```

You can access that dictionary directly as attrs:

```
[21]: tag_child.attrs
```

```
[21]: {'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.

```
[22]: tag_child.get('id')
```

```
[22]: 'boldest'
```

1.2.3 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 sting of the Tag object tag_child as follows:

```
[23]: tag_string=tag_child.string
      tag_string
```

```
[23]: 'Lebron James'
```

we can verify the type is Navigable String

```
[24]: type(tag_string)
```

```
[24]: 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 covert it to sting object in Python:

[25]:

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

[25]:

```
'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 Beautiful Soup will perform a match against that exact string. Consider the following HTML of rocket launchs:

[26]:

```
%%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:

[27]:

```
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>"
```

[28]:

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

1.3 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)

1.3.1 Name

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

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

```
[29]: [<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>]
```

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

```
[30]: first_row =table_rows[0]
      first_row
```

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

The type is tag

```
[31]: print(type(first_row))
```

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

we can obtain the child

```
[32]: first_row.td
```

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

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

```
[33]: 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.

[34]:
```python
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></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>
</td>
colunm 2 cell <td>80 kg</td>
```

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

[35]:
```python
list_input=table_bs .find_all(name=["tr", "td"])
list_input
```

[35]:
```
[<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></td><td>300
kg</td></tr>,
 <td>1</td>,
 <td><a href="https://en.wikipedia.org/wiki/Florida">Florida</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>
</td><td>80 kg</td></tr>,
 <td>3</td>,
 <td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a> </td>,
 <td>80 kg</td>]
```

1.4 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.

[36]:
```
table_bs.find_all(id="flight")
```

[36]:
```
[<td id="flight">Flight No</td>]
```

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

[37]:
```
list_input=table_bs.find_all(href="https://en.wikipedia.org/wiki/Florida")
list_input
```

[37]:
```
[<a href="https://en.wikipedia.org/wiki/Florida">Florida</a>,
 <a href="https://en.wikipedia.org/wiki/Florida">Florida</a>]
```

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

[38]:
```
table_bs.find_all(href=True)
```

[38]:
```
[<a href="https://en.wikipedia.org/wiki/Florida">Florida</a>,
 <a href="https://en.wikipedia.org/wiki/Texas">Texas</a>,
 <a href="https://en.wikipedia.org/wiki/Florida">Florida</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

```
table_bs.find_all('a', href=False)
```

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

[ ]:

```

```

Click here for the solution

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

1.4.1 string

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

[39]:

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

[39]:

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

1.5 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:

[40]:

```
%%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:

[41]:

```
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

[42]:

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

We can find the first table using the tag name table

[43]:
```
two_tables_bs.find("table")
```

[43]:
```
<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.

[44]:
```
two_tables_bs.find("table",class_='pizza')
```

[44]:
```
<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:

[45]:
```
url = "https://web.archive.org/web/20230224123642/https://www.ibm.com/us-en/"
```

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

[46]:
```
data  = requests.get(url).text
```

We create a BeautifulSoup object using the BeautifulSoup constructor

[47]:
```
soup = BeautifulSoup(data,"html.parser")  # create a soup object using the␣
 ↪variable 'data'
```

Scrape all links

[48]:
```
for link in soup.find_all('a',href=True):  # in html anchor/link is represented␣
 ↪by the tag <a>

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

```
https://web.archive.org/web/20230224123642/https://www.ibm.com/reports/threat-
intelligence/
https://web.archive.org/web/20230224123642/https://www.ibm.com/about
https://web.archive.org/web/20230224123642/https://www.ibm.com/consulting/?lnk=f
lathl
https://web.archive.org/web/20230224123642/https://www.ibm.com/consulting/strate
gy/?lnk=flathl
https://web.archive.org/web/20230224123642/https://www.ibm.com/consulting/ibmix?
lnk=flathl
https://web.archive.org/web/20230224123642/https://www.ibm.com/consulting/techno
logy/
```

```
https://web.archive.org/web/20230224123642/https://www.ibm.com/consulting/operat
ions/?lnk=flathl
https://web.archive.org/web/20230224123642/https://www.ibm.com/strategic-
partnerships
https://web.archive.org/web/20230224123642/https://www.ibm.com/employment/?lnk=f
latitem
https://web.archive.org/web/20230224123642/https://www.ibm.com/impact
https://web.archive.org/web/20230224123642/https://research.ibm.com/
https://web.archive.org/web/20230224123642/https://www.ibm.com/
```

1.6 Scrape all images Tags

[49]:

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

```
<img alt="Person standing with arms crossed" aria-describedby="bx--image-1"
class="bx--image__img" src="https://web.archive.org/web/20230224123642im_/https:
//1.dam.s81c.com/p/0a23e414312bcb6f/08196d0e04260ae5_cropped.jpg.global.sr_16x9.
jpg"/>
https://web.archive.org/web/20230224123642im_/https://1.dam.s81c.com/p/0a23e4143
12bcb6f/08196d0e04260ae5_cropped.jpg.global.sr_16x9.jpg
<img alt="Team members at work in a conference room" aria-describedby="bx--
image-2" class="bx--image__img" src="https://web.archive.org/web/20230224123642i
m_/https://1.dam.s81c.com/p/06655c075aa3aa29/CaitOppermann_2019_12_06_IBMGarage_
DSC3304.jpg.global.m_16x9.jpg"/>
https://web.archive.org/web/20230224123642im_/https://1.dam.s81c.com/p/06655c075
aa3aa29/CaitOppermann_2019_12_06_IBMGarage_DSC3304.jpg.global.m_16x9.jpg
<img alt="Coworkers looking at laptops" aria-describedby="bx--image-3" class="bx
--image__img" src="https://web.archive.org/web/20230224123642im_/https://1.dam.s
81c.com/p/08f951353c2707b8/052022_CaitOppermann_InsideIBM_London_2945_03.jpg.glo
bal.sr_16x9.jpg"/>
https://web.archive.org/web/20230224123642im_/https://1.dam.s81c.com/p/08f951353
c2707b8/052022_CaitOppermann_InsideIBM_London_2945_03.jpg.global.sr_16x9.jpg
<img alt="Cloud developer with red sweater coding at desk" aria-describedby="bx
--image-4" class="bx--image__img" src="https://web.archive.org/web/2023022412364
2im_/https://1.dam.s81c.com/p/064e0139f5a3aa5e/0500002_Lowell_LI_100119.jpg.glob
al.sr_16x9.jpg"/>
https://web.archive.org/web/20230224123642im_/https://1.dam.s81c.com/p/064e0139f
5a3aa5e/0500002_Lowell_LI_100119.jpg.global.sr_16x9.jpg
<img alt="Aerial view of automated conveyer belt and machinery at work" aria-
describedby="bx--image-5" class="bx--image__img" src="https://web.archive.org/we
b/20230224123642im_/https://1.dam.s81c.com/p/0795cae91a25156f/conveyorrobottopvi
ew.jpg.global.sr_16x9.jpg"/>
https://web.archive.org/web/20230224123642im_/https://1.dam.s81c.com/p/0795cae91
a25156f/conveyorrobottopview.jpg.global.sr_16x9.jpg
<img alt="Overhead view of partners collaborating on design with laptops and
coffee" aria-describedby="bx--image-6" class="bx--image__img" src="https://web.a
```

```
rchive.org/web/20230224123642im_/https://1.dam.s81c.com/p/06dfa9ccdba4ec79/1f417
900-9042-44d1-9c219a854bbb62ea.jpg.global.sr_16x9.jpg"/>
https://web.archive.org/web/20230224123642im_/https://1.dam.s81c.com/p/06dfa9ccd
ba4ec79/1f417900-9042-44d1-9c219a854bbb62ea.jpg.global.sr_16x9.jpg
```

1.7 Scrape data from HTML tables

[50]:

```python
#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 are there in the color table.

[51]:

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

[52]:

```python
soup = BeautifulSoup(data,"html.parser")
```

[53]:

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

[54]:

```python
#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))
```

```
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--->#FFEFD5
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
```

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

[55]:
```
import pandas as pd
```

[56]:
```
#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.

[57]:
```
# get the contents of the webpage in text format and store in a variable called␣
  ↪data
data  = requests.get(url).text
```

[58]:
```
soup = BeautifulSoup(data,"html.parser")
```

[59]:
```
#find all html tables in the web page
tables = soup.find_all('table') # in html table is represented by the tag␣
  ↪<table>
```

[60]:
```
# we can see how many tables were found by checking the length of the tables␣
  ↪list
len(tables)
```

[60]:
```
0
```

Assume that we are looking for the `10 most densly populated countries` table, we can look through the tables list and find the right one we are look 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.

[69]:

```
table_index = None  # default if nothing is found

for index, table in enumerate(tables):
    if "10 most densely populated countries" in str(table):
        table_index = index
        break   # stop at first match (optional)

print(table_index)  # will print None if not found
```

```
None
```

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

[71]:

```
table_index = None

for index, table in enumerate(tables):
    # case-insensitive search to avoid missing due to capitalization/spelling
    if "densely populated countries" in str(table).lower():
        table_index = index
        break

if table_index is not None:
    print(tables[table_index].prettify())
else:
    print("Table not found")
```

```
Table not found
```

[ ]:

```

```

[98]:

```
!pip install lxml
import requests
import pandas as pd

url = "https://en.wikipedia.org/wiki/
 ↪List_of_countries_and_dependencies_by_population_density"

# fetch HTML manually
headers = {"User-Agent": "Data-Analytics-Lab/1.0 (https://example.com/contact)"}
response = requests.get(url, headers=headers)



# now let pandas parse the HTML string
tables = pd.read_html(response.text)
```

```python
print(len(tables))   # how many tables did we get?

# usually the first table is the "10 most densely populated countries"
population_data = tables[0]
print(population_data.head())

population_density_top10 = tables[1]
print(population_density_top10.head(10))


print(population_density_top10.dtypes)
print(population_density_top10.head())
```

```
Requirement already satisfied: lxml in /opt/conda/lib/python3.12/site-packages
(6.0.1)
2
            Location  Density          Population    Land area               \
            Location     /km2     /mi2 Population          km2          mi2
0              World     55.0    140.0 8231613070  148940000.0  57510000.00
1      Macau (China)  22000.0  57000.0     720262         33.0        13.00
2             Monaco  19000.0  49000.0      38631          2.0         0.77
3          Singapore   8120.0  21000.0    5832387        718.0       277.00
4  Hong Kong (China)   7062.0  18290.0    7414910       1050.0       410.00

  Ref.
  Ref.
0  [b]
1  NaN
2  NaN
3  NaN
4  NaN
  vteLists of countries by population statistics  \
0                                          Global
1                            Continents/subregions
2                                Intercontinental
3                              Cities/urban areas
4                                 Past and future
5                              Population density
6                               Growth indicators
7                                 Life expectancy
8                              Other demographics
9                                          Health

    vteLists of countries by population statistics.1
0  Current population United Nations Demographics…
1  Africa Antarctica Asia Europe North America Ca…
2  Americas Arab world Commonwealth of Nations Eu…
3  World cities National capitals Megacities Mega…
```

```
4  Past and future population Estimates of histor…
5  Current density Past and future population den…
6  Population growth rate Natural increase Net re…
7  World Africa Asia Europe North America Oceania…
8  Age at childbearing Age at first marriage Age …
9  Antidepressant consumption Antiviral medicatio…
vteLists of countries by population statistics      object
vteLists of countries by population statistics.1    object
dtype: object
  vteLists of countries by population statistics  \
0                                         Global
1                          Continents/subregions
2                               Intercontinental
3                            Cities/urban areas
4                                Past and future

    vteLists of countries by population statistics.1
0  Current population United Nations Demographics…
1  Africa Antarctica Asia Europe North America Ca…
2  Americas Arab world Commonwealth of Nations Eu…
3  World cities National capitals Megacities Mega…
4  Past and future population Estimates of histor…
```

1.9 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`.

[102]:

```python
for i, t in enumerate(tables):
    try:
        temp = pd.read_html(str(t), flavor="bs4")[0]
        print(f"\n--- Table {i} ---")
        print(temp.head(3))
    except:
        pass
```

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

[104]:

```python
print(len(tables))
```

```
2
```

[105]:

```
for i, tbl in enumerate(tables):
    print(f"\n--- Table {i} ---")
    print(tbl.head())
```

```
--- Table 0 ---
            Location  Density           Population    Land area               \
            Location     /km2     /mi2  Population          km2          mi2   
0              World     55.0    140.0  8231613070  148940000.0  57510000.00   
1      Macau (China)  22000.0  57000.0      720262         33.0        13.00   
2             Monaco  19000.0  49000.0       38631          2.0         0.77   
3          Singapore   8120.0  21000.0     5832387        718.0       277.00   
4  Hong Kong (China)   7062.0  18290.0     7414910       1050.0       410.00   

  Ref.  
  Ref.  
0  [b]  
1  NaN  
2  NaN  
3  NaN  
4  NaN  

--- Table 1 ---
  vteLists of countries by population statistics  \
0                                            Global   
1                             Continents/subregions   
2                                  Intercontinental   
3                                Cities/urban areas   
4                                   Past and future   

    vteLists of countries by population statistics.1  
0  Current population United Nations Demographics…  
1  Africa Antarctica Asia Europe North America Ca…  
2  Americas Arab world Commonwealth of Nations Eu…  
3  World cities National capitals Megacities Mega…  
4  Past and future population Estimates of histor…  
```

[107]:

```
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)

dataframe_list = pd.read_html(response.text, flavor='bs4')
```

[109]:

```
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)

dataframe_list = pd.read_html(response.text, flavor='bs4')
```

1.10 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`.

[111]:
```
import pandas as pd
import requests

url = "https://en.wikipedia.org/wiki/
 ↪List_of_countries_and_dependencies_by_population_density"

# Add headers so Wikipedia doesn't block the request
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)

# Now parse directly
dataframe_list = pd.read_html(response.text, flavor='bs4')

print(len(dataframe_list))   # how many tables did we get?
print(dataframe_list[0].head())  # first table is the population density one
```

```
2
            Location  Density           Population    Land area               \
            Location     /km2     /mi2  Population          km2          mi2
0              World     55.0    140.0  8231613070  148940000.0  57510000.00
1      Macau (China)  22000.0  57000.0      720262         33.0        13.00
2             Monaco  19000.0  49000.0       38631          2.0         0.77
3          Singapore   8120.0  21000.0     5832387        718.0       277.00
4  Hong Kong (China)   7062.0  18290.0     7414910       1050.0       410.00

  Ref.
  Ref.
0  [b]
1  NaN
2  NaN
3  NaN
4  NaN
```

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

[112]:
```
len(dataframe_list)
```

[112]: 2

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

[114]:
```
# check how many tables were scraped
print(len(dataframe_list))

# table 0 is the one you want
```

```
population_data = dataframe_list[0]
print(population_data.head())
```

```
2
            Location  Density           Population    Land area               \
            Location     /km2     /mi2  Population          km2          mi2
0              World     55.0    140.0  8231613070  148940000.0  57510000.00
1      Macau (China)  22000.0  57000.0      720262         33.0        13.00
2             Monaco  19000.0  49000.0       38631          2.0         0.77
3          Singapore   8120.0  21000.0     5832387        718.0       277.00
4  Hong Kong (China)   7062.0  18290.0     7414910       1050.0       410.00

  Ref.
  Ref.
0  [b]
1  NaN
2  NaN
3  NaN
4  NaN
```

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.

[116]:

```
import requests
import pandas as pd

url = "https://en.wikipedia.org/wiki/
 ↪List_of_countries_and_dependencies_by_population_density"

# Add headers so Wikipedia doesn't block us
headers = {"User-Agent": "Mozilla/5.0"}
response = requests.get(url, headers=headers)

# Parse the tables from the HTML string
tables = pd.read_html(response.text, match="Location", flavor="bs4")

df = tables[0]

# Clean up the headers
df.columns = ["Location", "Density (/km2)", "Density (/mi2)",
              "Population", "Land area (km2)", "Land area (mi2)", "Ref."]

# Drop the extra header row inside the table
df = df.drop(0).reset_index(drop=True)

print(df.head(10))
```

```
            Location  Density (/km2)  Density (/mi2)  Population  \
0      Macau (China)         22000.0         57000.0      720262
```

```
1           Monaco         19000.0         49000.0      38631
2        Singapore          8120.0         21000.0    5832387
3  Hong Kong (China)         7062.0         18290.0    7414910
4     Gibraltar (UK)          5800.0         15000.0      39329
5          Bahrain          2034.0          5270.0    1607049
6       Vatican City          1800.0          4700.0        882
7         Maldives          1759.0          4560.0     527799
8            Malta          1686.0          4370.0     539607
9       Bangladesh          1333.0          3450.0  173562364

   Land area (km2)  Land area (mi2) Ref.
0             33.0            13.00  NaN
1              2.0             0.77  NaN
2            718.0           277.00  NaN
3           1050.0           410.00  NaN
4              6.8             2.60  [c]
5            790.0           310.00  NaN
6              0.5             0.19  [d]
7            300.0           120.00  NaN
8            320.0           120.00  NaN
9         130170.0         50260.00  NaN
```

1.11 Authors

Ramesh Sannareddy

1.11.1 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 | | –!>