Web Scraping Lab

Estimated time needed: **30** minutes

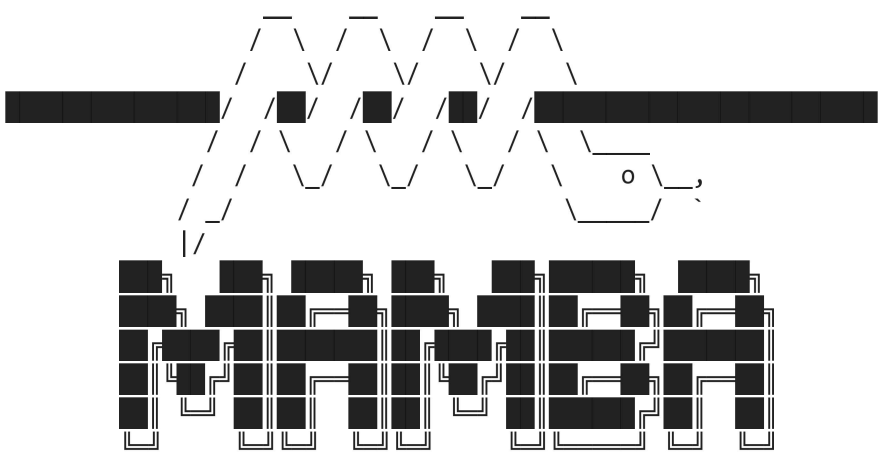
Objectives

After completing this lab you will be able to:

Table of Contents

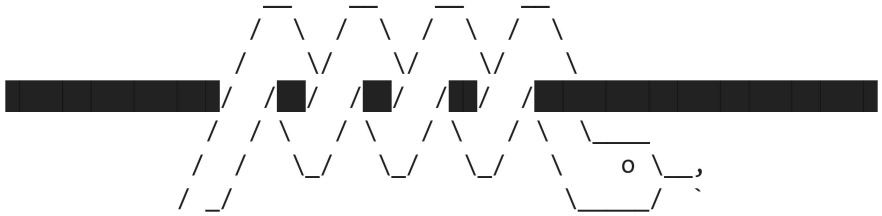
For this lab, we are going to be using Python and several Python libraries. Some of these libraries might be installed in your lab environment or in SN Labs. Others may need to be installed by you. The cells below will install these libraries when executed.

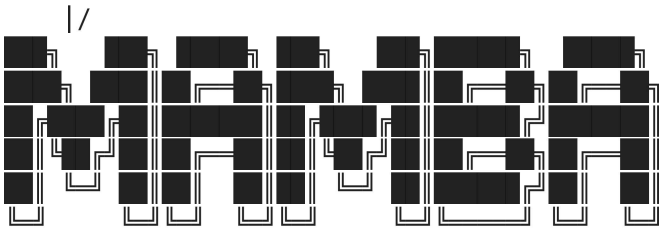
In [1]:
```
!mamba install bs4==4.10.0 -y
!pip install lxml==4.6.4
!mamba install html5lib==1.1 -y
# !pip install requests==2.26.0
```



```
mamba (1.4.2) supported by @QuantStack

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

```
Looking for: ['bs4==4.10.0']

[+] 0.0s
[+] 0.1s
pkgs/main/linux-64 ──────────╸                   0.0 B /  ??.?MB @  ??.?MB/s  0.1s
pkgs/main/noarch   ──────────╸                   0.0 B /  ??.?MB @  ??.?MB/s  0.1s
pkgs/r/linux-64    ──────────╸                   0.0 B /  ??.?MB @  ??.?MB/s  0.1s
pkgs/r/noarch      ──────────╸                   0.0 B /  ??.?MB @  ??.?MB/s  0.1spkg
s/r/linux-64                                                 No change
pkgs/main/noarch                                                No change
pkgs/r/noarch                                                   No change
pkgs/main/linux-64                                              No change

Pinned packages:
  - python 3.7.*


Transaction

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

  All requested packages already installed

Requirement already satisfied: lxml==4.6.4 in /home/jupyterlab/conda/envs/python/li
b/python3.7/site-packages (4.6.4)
```





```
mamba (1.4.2) supported by @QuantStack

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

```
Looking for: ['html5lib==1.1']

pkgs/main/linux-64                                      Using cache
pkgs/main/noarch                                        Using cache
pkgs/r/linux-64                                         Using cache
pkgs/r/noarch                                           Using cache

Pinned packages:
  - python 3.7.*


Transaction

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

  All requested packages already installed
```

Import the required modules and functions

In [2]:
```
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 [3]:
```
%%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 [4]:
```
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 [5]:
```
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 [5]:
```
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 [6]:
```
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 [7]:
```
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 [8]:
```
tag_object=soup.h3
tag_object
```

Out[8]:
```
<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 [9]:
```
tag_child =tag_object.b
tag_child
```

Out[9]:
```
<b id="boldest">Lebron James</b>
```

You can access the parent with the `parent`

In [10]:
```
parent_tag=tag_child.parent
parent_tag
```

Out[10]:
```
<h3><b id="boldest">Lebron James</b></h3>
```

this is identical to

In [11]:
```
tag_object
```

Out[11]:
```
<h3><b id="boldest">Lebron James</b></h3>
```

`tag_object` parent is the `body` element.

In [13]:
```
tag_object.parent
```

Out[13]:
```
<body><h3><b id="boldest">Lebron James</b></h3><p> Salary: $ 92,000,000 </p><h3> S
tephen 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 [14]:
```
sibling_1=tag_object.next_sibling
sibling_1
```

Out[14]:
```
<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 [15]:
```
sibling_2=sibling_1.next_sibling
sibling_2
```

Out[15]:
```
<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 [16]:
```
sibling_2.next_sibling
```

Out[16]:
```
<p> Salary: $85,000, 000 </p>
```

▶ 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 [17]:
```
tag_child['id']
```

Out[17]:
```
'boldest'
```

You can access that dictionary directly as `attrs`:

In [18]:
```
tag_child.attrs
```

Out[18]:
```
{'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 [19]:
```
tag_child.get('id')
```

Out[19]:
```
'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 sting of the `Tag` object `tag_child` as follows:

In [20]:
```
tag_string=tag_child.string
tag_string
```

Out[20]: 'Lebron James'

we can verify the type is Navigable String

In [21]:
```
type(tag_string)
```

Out[21]: 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:

In [22]:
```
unicode_string = str(tag_string)
unicode_string
```

Out[22]: '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:

In [23]:
```
%%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>
```

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 [24]:
```
table="<table><tr><td id='flight'>Flight No</td><td>Launch site</td> <td>Payload ma
```

In [25]:
```
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 [26]:
```
table_rows=table_bs.find_all('tr')
table_rows
```

Out[26]:
```
[<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><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 [27]:
```
first_row =table_rows[0]
first_row
```

Out[27]:
```
<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>
```

The type is `tag`

In [28]:
```
print(type(first_row))
```

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

we can obtain the child

In [29]:
```
first_row.td
```

Out[29]:
```
<td id="flight">Flight No</td>
```

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

In [30]:
```
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><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
> </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.

In [31]:
```
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.

In [32]:
```
list_input=table_bs .find_all(name=["tr", "td"])
list_input
```

Out[32]:
```
[<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 [33]:
```
table_bs.find_all(id="flight")
```

Out[33]:
```
[<td id="flight">Flight No</td>]
```

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

In [34]:
```
list_input=table_bs.find_all(href="https://en.wikipedia.org/wiki/Florida")
list_input
```

Out[34]:
```
[<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 [35]:
```
table_bs.find_all(href=True)
```

Out[35]:
```
[<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

Exercise: `find_all`

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

In [36]:
```
table_bs.find_all(href=False)
```

Out[36]:
```
[<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><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></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><td>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>]
```

▶ Click here for the solution

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

In [38]:
```
soup.find_all(id="boldest")
```

Out[38]:
```
[<b id="boldest">Lebron James</b>]
```

▶ 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 [39]:
```
table_bs.find_all(string="Florida")
```

Out[39]:
```
['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 [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>
```

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 [41]:
```
two_tables="<h3>Rocket Launch </h3><p><table class='rocket'><tr><td>F
light 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>Piz
za Place</td><td>Orders</td> <td>Slices </td></tr><tr><td>Domino's Pi
zza</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`

In [42]:
```
two_tables_bs= BeautifulSoup(two_tables, 'html.parser')
```

We can find the first table using the tag name table

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

```
<table class="rocket"><tr><td>Flight No</td><td>Launch site</td> <td>Payload mas
s</td></tr><tr><td>1</td><td>Florida</td><td>300 kg</td></tr><tr><td>2</td><td>T
exas</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 [44]:
```
two_tables_bs.find("table",class_='pizza')
```

```
<table class="pizza"><tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></t
r><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 [46]:
```
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 [47]:
```
data  = requests.get(url).text
```

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

In [48]:
```
soup = BeautifulSoup(data,"html.parser")  # create a soup object using the variable 'data'
```

Scrape all links

In [49]:
```
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/consulting/ibmix
https://www.ibm.com/community/ibm-techxchange-conference
https://www.ibm.com/products/watsonx-ai
https://www.ibm.com/products/planning-analytics?lnk=flatitem
https://www.ibm.com/products/spss-statistics/pricing
https://www.ibm.com/resources/the-data-differentiator/scale-ai
https://www.ibm.com/cloud?lnk=flatitem
https://www.ibm.com/products
#bx--custom-footnotes
https://www.ibm.com/consulting
https://www.ibm.com/about
https://www.gartner.com/en/documents/4007140
https://www.ibm.com/
```

Scrape all images Tags

In [50]:
```
for link in soup.find_all('img'):# in html image is represented by the tag <img>
    print(link)
    print(link.get('src'))
```

```
<img alt="Concentric illustration showing watsonx.ai capabilities" aria-describe
dby="bx--image-1" class="bx--image__img" src="https://1.dam.s81c.com/p/0b52089e9
94c221c/ibm-watsonxai-aspectratio-2-by-1.png.global.xs_1x1.png"/>
https://1.dam.s81c.com/p/0b52089e994c221c/ibm-watsonxai-aspectratio-2-by-1.png.g
lobal.xs_1x1.png
<img alt="Illustration of workers digital planning with blue dots and post-its"
aria-describedby="bx--image-2" class="bx--image__img" src="https://1.dam.s81c.co
m/p/0b5258b33acc8e04/homepage-planning-analytics-card.png.global.xs_1x1.png"/>
https://1.dam.s81c.com/p/0b5258b33acc8e04/homepage-planning-analytics-card.png.g
lobal.xs_1x1.png
<img alt="Illustration of people at large digital dashboard" aria-describedby="b
x--image-3" class="bx--image__img" src="https://1.dam.s81c.com/p/0b5258b292cc8c3
c/ibm-SPSS-home-card.png.global.xs_1x1.png"/>
https://1.dam.s81c.com/p/0b5258b292cc8c3c/ibm-SPSS-home-card.png.global.xs_1x1.p
ng
<img alt="Dots and lines illustration" aria-describedby="bx--image-4" class="bx-
-image__img" src="https://1.dam.s81c.com/p/08a5a9b9c2464461/01-Your_guide_to_dif
ferentiating_with_data_568x320.jpg.global.xs_1x1.png"/>
https://1.dam.s81c.com/p/08a5a9b9c2464461/01-Your_guide_to_differentiating_with_
data_568x320.jpg.global.xs_1x1.png
<img alt="Grid of IBM consultant portraits" class="bx--image__img" src="https://
1.dam.s81c.com/p/0aac9cf57bcbf324/dotcom-1-overview.jpg"/>
https://1.dam.s81c.com/p/0aac9cf57bcbf324/dotcom-1-overview.jpg
```

Scrape data from HTML tables

In [51]:
```
#The below url contains an html table with data about colors and colo
r codes.
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c
loud/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.

In [52]:
```
# get the contents of the webpage in text format and store in a varia
ble called data
data  = requests.get(url).text
```

In [53]:
```
soup = BeautifulSoup(data,"html.parser")
```

In [54]:
```
#find a html table in the web page
table = soup.find('table') # in html table is represented by the tag
<table>
```

In [55]:
```
#Get all rows from the table
for row in table.find_all('tr'): # in html table row is represented b
y the tag <tr>
    # Get all columns in each row.
    cols = row.find_all('td') # in html a column is represented by th
e tag <td>
    color_name = cols[2].string # store the value in column 3 as colo
```

```
r_name
    color_code = cols[3].string # store the value in column 4 as colo
r_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
```

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

In [56]:
```
import pandas as pd
```

In [57]:
```
#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 [58]:
```
# get the contents of the webpage in text format and store in a varia
ble called data
```

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

In [59]:
```
soup = BeautifulSoup(data,"html.parser")
```

In [60]:
```
#find all html tables in the web page
tables = soup.find_all('table') # in html table is represented by the tag <table>
```

In [61]:
```
# we can see how many tables were found by checking the length of the tables list
len(tables)
```

```
29
```

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.

In [62]:
```
for index,table in enumerate(tables):
    if ("10 most densely populated countries" in str(table)):
        table_index = index
print(table_index)
```

```
7
```

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

In [63]:
```
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_106-0">
   <a href="#cite_note-:10-106">
    [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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="90
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/th
umb/4/48/Flag_of_Singapore.svg/23px-Flag_of_Singapore.svg.png" srcset="//upload.
wikimedia.org/wikipedia/commons/thumb/4/48/Flag_of_Singapore.svg/35px-Flag_of_Si
```

```
ngapore.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/4/48/Flag_o
f_Singapore.svg/45px-Flag_of_Singapore.svg.png 2x" width="23"/>
    </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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="100
0" decoding="async" height="14" src="//upload.wikimedia.org/wikipedia/commons/th
umb/f/f9/Flag_of_Bangladesh.svg/23px-Flag_of_Bangladesh.svg.png" srcset="//uploa
d.wikimedia.org/wikipedia/commons/thumb/f/f9/Flag_of_Bangladesh.svg/35px-Flag_of
_Bangladesh.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/f/f9/Fl
ag_of_Bangladesh.svg/46px-Flag_of_Bangladesh.svg.png 2x" width="23"/>
    </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">
      <img alt="" class="thumbborder" data-file-height="216" data-file-width="43
2" decoding="async" height="12" src="//upload.wikimedia.org/wikipedia/commons/th
umb/0/00/Flag_of_Palestine.svg/23px-Flag_of_Palestine.svg.png" srcset="//upload.
wikimedia.org/wikipedia/commons/thumb/0/00/Flag_of_Palestine.svg/35px-Flag_of_Pa
lestine.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/0/00/Flag_o
```

```
f_Palestine.svg/46px-Flag_of_Palestine.svg.png 2x" width="23"/>
     </span>
     <a href="/wiki/State_of_Palestine" title="State of Palestine">
      Palestine
     </a>
     <sup class="reference" id="cite_ref-107">
      <a href="#cite_note-107">
       [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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="90
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/th
umb/7/72/Flag_of_the_Republic_of_China.svg/23px-Flag_of_the_Republic_of_China.sv
g.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_R
epublic_of_China.svg/35px-Flag_of_the_Republic_of_China.svg.png 1.5x, //upload.w
ikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/45px
-Flag_of_the_Republic_of_China.svg.png 2x" width="23"/>
    </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">
```

```
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="90
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/th
umb/0/09/Flag_of_South_Korea.svg/23px-Flag_of_South_Korea.svg.png" srcset="//upl
oad.wikimedia.org/wikipedia/commons/thumb/0/09/Flag_of_South_Korea.svg/35px-Flag
_of_South_Korea.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/0/0
9/Flag_of_South_Korea.svg/45px-Flag_of_South_Korea.svg.png 2x" width="23"/>
    </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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="90
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/th
umb/5/59/Flag_of_Lebanon.svg/23px-Flag_of_Lebanon.svg.png" srcset="//upload.wiki
media.org/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/35px-Flag_of_Lebanon.
svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/5/59/Flag_of_Lebano
n.svg/45px-Flag_of_Lebanon.svg.png 2x" width="23"/>
    </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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="90
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/th
```

```
umb/1/17/Flag_of_Rwanda.svg/23px-Flag_of_Rwanda.svg.png" srcset="//upload.wikime
dia.org/wikipedia/commons/thumb/1/17/Flag_of_Rwanda.svg/35px-Flag_of_Rwanda.svg.
png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/1/17/Flag_of_Rwanda.sv
g/45px-Flag_of_Rwanda.svg.png 2x" width="23"/>
    </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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="100
0" decoding="async" height="14" src="//upload.wikimedia.org/wikipedia/commons/th
umb/5/50/Flag_of_Burundi.svg/23px-Flag_of_Burundi.svg.png" srcset="//upload.wiki
media.org/wikipedia/commons/thumb/5/50/Flag_of_Burundi.svg/35px-Flag_of_Burundi.
svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/5/50/Flag_of_Burund
i.svg/46px-Flag_of_Burundi.svg.png 2x" width="23"/>
    </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">
     <img alt="" class="thumbborder" data-file-height="900" data-file-width="135
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/en/thumb/
4/41/Flag_of_India.svg/23px-Flag_of_India.svg.png" srcset="//upload.wikimedia.or
g/wikipedia/en/thumb/4/41/Flag_of_India.svg/35px-Flag_of_India.svg.png 1.5x, //u
```

```
pload.wikimedia.org/wikipedia/en/thumb/4/41/Flag_of_India.svg/45px-Flag_of_Indi
a.svg.png 2x" width="23"/>
    </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">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="90
0" decoding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/th
umb/2/20/Flag_of_the_Netherlands.svg/23px-Flag_of_the_Netherlands.svg.png" srcse
t="//upload.wikimedia.org/wikipedia/commons/thumb/2/20/Flag_of_the_Netherlands.s
vg/35px-Flag_of_the_Netherlands.svg.png 1.5x, //upload.wikimedia.org/wikipedia/c
ommons/thumb/2/20/Flag_of_the_Netherlands.svg/45px-Flag_of_the_Netherlands.svg.p
ng 2x" width="23"/>
    </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 [64]:
```
population_data = pd.DataFrame(columns=["Rank", "Country", "Populatio
n", "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, "Count
ry":country, "Population":population, "Area":area, "Density":densit
y}, 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 [65]:
```
pd.read_html(str(tables[5]), flavor='bs4')
```

Out[65]: [

	#	\
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
	Graphs are temporarily unavailable due to technical issues.	
0	NaN	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	
11	NaN	
12	Notes: .mw-parser-output .reflist{font-size:90...	

	Most populous countries	\
	Unnamed: 1_level_1	
	Unnamed: 1_level_2	
	Unnamed: 1_level_3	
	Unnamed: 1_level_4	
	Unnamed: 1_level_5	
	Unnamed: 1_level_6	
	Unnamed: 1_level_7	
	Unnamed: 1_level_8	
	Unnamed: 1_level_9	
	Unnamed: 1_level_10	
	Unnamed: 1_level_11	
0	Graphs are temporarily unavailable due to tech...	
1	China[B]	
2	India	
3	United States	
4	Indonesia	
5	Pakistan	
6	Brazil	
7	Nigeria	
8	Bangladesh	
9	Russia	
10	Mexico	
11	World total	
12	Notes: .mw-parser-output .reflist{font-size:90...	

	2000	\
	Unnamed: 2_level_1	
	Unnamed: 2_level_2	
	Unnamed: 2_level_3	

```
                                    Unnamed: 2_level_4
                                    Unnamed: 2_level_5
                                    Unnamed: 2_level_6
                                    Unnamed: 2_level_7
                                    Unnamed: 2_level_8
                                    Unnamed: 2_level_9
                                   Unnamed: 2_level_10
                                   Unnamed: 2_level_11
0                                                  NaN
1                                                 1270
2                                                 1053
3                                                  283
4                                                  212
5                                                  136
6                                                  176
7                                                  123
8                                                  131
9                                                  146
10                                                 103
11                                                6127
12  Notes: .mw-parser-output .reflist{font-size:90...

                                                  2015  \
                                    Unnamed: 3_level_1
                                    Unnamed: 3_level_2
                                    Unnamed: 3_level_3
                                    Unnamed: 3_level_4
                                    Unnamed: 3_level_5
                                    Unnamed: 3_level_6
                                    Unnamed: 3_level_7
                                    Unnamed: 3_level_8
                                    Unnamed: 3_level_9
                                   Unnamed: 3_level_10
                                   Unnamed: 3_level_11
0                                                  NaN
1                                                 1376
2                                                 1311
3                                                  322
4                                                  258
5                                                  208
6                                                  206
7                                                  182
8                                                  161
9                                                  146
10                                                 127
11                                                7349
12  Notes: .mw-parser-output .reflist{font-size:90...

                                                2030[A]  \
                                    Unnamed: 4_level_1
                                    Unnamed: 4_level_2
                                    Unnamed: 4_level_3
                                    Unnamed: 4_level_4
                                    Unnamed: 4_level_5
                                    Unnamed: 4_level_6
                                    Unnamed: 4_level_7
```

```
                                  Unnamed: 4_level_8
                                  Unnamed: 4_level_9
                                 Unnamed: 4_level_10
                                 Unnamed: 4_level_11
0                                                NaN
1                                               1416
2                                               1528
3                                                356
4                                                295
5                                                245
6                                                228
7                                                263
8                                                186
9                                                149
10                                               148
11                                              8501
12  Notes: .mw-parser-output .reflist{font-size:90...

   Graphs are temporarily unavailable due to technical issues.
                                           Unnamed: 5_level_1
                                           Unnamed: 5_level_2
                                           Unnamed: 5_level_3
                                           Unnamed: 5_level_4
                                           Unnamed: 5_level_5
                                           Unnamed: 5_level_6
                                           Unnamed: 5_level_7
                                           Unnamed: 5_level_8
                                           Unnamed: 5_level_9
                                          Unnamed: 5_level_10
                                          Unnamed: 5_level_11
0                                                        NaN
1                                                        NaN
2                                                        NaN
3                                                        NaN
4                                                        NaN
5                                                        NaN
6                                                        NaN
7                                                        NaN
8                                                        NaN
9                                                        NaN
10                                                       NaN
11                                                       NaN
12  Notes: .mw-parser-output .reflist{font-size:90...           ,
    0                                                    1
0 NaN  Graphs are temporarily unavailable due to tech...]
```

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

In [66]:
```
population_data_read_html = pd.read_html(str(tables[5]), flavor='bs4')[0]

population_data_read_html
```

Out[66]:

#	Most populous countries	2000	2015	2030[A]	Graphs are temporarily unavailable due to technical issues.
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_1	Unnamed: 2_level_1	Unnamed: 3_level_1	Unnamed: 4_level_1	Unnamed: 5_level_1
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_2	Unnamed: 2_level_2	Unnamed: 3_level_2	Unnamed: 4_level_2	Unnamed: 5_level_2
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_3	Unnamed: 2_level_3	Unnamed: 3_level_3	Unnamed: 4_level_3	Unnamed: 5_level_3
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_4	Unnamed: 2_level_4	Unnamed: 3_level_4	Unnamed: 4_level_4	Unnamed: 5_level_4
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_5	Unnamed: 2_level_5	Unnamed: 3_level_5	Unnamed: 4_level_5	Unnamed: 5_level_5
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_6	Unnamed: 2_level_6	Unnamed: 3_level_6	Unnamed: 4_level_6	Unnamed: 5_level_6
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_7	Unnamed: 2_level_7	Unnamed: 3_level_7	Unnamed: 4_level_7	Unnamed: 5_level_7

	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_8	Unnamed: 2_level_8	Unnamed: 3_level_8	Unnamed: 4_level_8	Unnamed: 5_level_8
	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_9	Unnamed: 2_level_9	Unnamed: 3_level_9	Unnamed: 4_level_9	Unnamed: 5_level_9
	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_10	Unnamed: 2_level_10	Unnamed: 3_level_10	Unnamed: 4_level_10	Unnamed: 5_level_10
	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_11	Unnamed: 2_level_11	Unnamed: 3_level_11	Unnamed: 4_level_11	Unnamed: 5_level_11
0	NaN	Graphs are temporarily unavailable due to tech...	NaN	NaN	NaN	NaN
1	1	China[B]	1270	1376	1416	NaN
2	2	India	1053	1311	1528	NaN
3	3	United States	283	322	356	NaN
4	4	Indonesia	212	258	295	NaN
5	5	Pakistan	136	208	245	NaN
6	6	Brazil	176	206	228	NaN
7	7	Nigeria	123	182	263	NaN
8	8	Bangladesh	131	161	186	NaN
9	9	Russia	146	146	149	NaN
10	10	Mexico	103	127	148	NaN
11	NaN	World total	6127	7349	8501	NaN
12	Notes: .mw-parser-output .reflist{font-size:90...	Notes: .mw-parser-output	Notes: .mw-parser-output	Notes: .mw-parser-output	Notes: .mw-parser-output	Notes: .mw-parser-output .reflist{font-size:90...

.reflist{font-size:90... .reflist{font-size:90... .reflist{font-size:90... .reflist{font-size:90...

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 [67]:
```
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 [68]:
```
len(dataframe_list)
```

Out[68]: 26

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

In [69]:
```
dataframe_list[5]
```

Out[69]:

#	Most populous countries	2000	2015	2030[A]	Graphs are temporarily unavailable due to technical issues.
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_1	Unnamed: 2_level_1	Unnamed: 3_level_1	Unnamed: 4_level_1	Unnamed: 5_level_1
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_2	Unnamed: 2_level_2	Unnamed: 3_level_2	Unnamed: 4_level_2	Unnamed: 5_level_2
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_3	Unnamed: 2_level_3	Unnamed: 3_level_3	Unnamed: 4_level_3	Unnamed: 5_level_3
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_4	Unnamed: 2_level_4	Unnamed: 3_level_4	Unnamed: 4_level_4	Unnamed: 5_level_4
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_5	Unnamed: 2_level_5	Unnamed: 3_level_5	Unnamed: 4_level_5	Unnamed: 5_level_5
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_6	Unnamed: 2_level_6	Unnamed: 3_level_6	Unnamed: 4_level_6	Unnamed: 5_level_6
Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_7	Unnamed: 2_level_7	Unnamed: 3_level_7	Unnamed: 4_level_7	Unnamed: 5_level_7

	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_8	Unnamed: 2_level_8	Unnamed: 3_level_8	Unnamed: 4_level_8	Unnamed: 5_level_8
	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_9	Unnamed: 2_level_9	Unnamed: 3_level_9	Unnamed: 4_level_9	Unnamed: 5_level_9
	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_10	Unnamed: 2_level_10	Unnamed: 3_level_10	Unnamed: 4_level_10	Unnamed: 5_level_10
	Graphs are temporarily unavailable due to technical issues.	Unnamed: 1_level_11	Unnamed: 2_level_11	Unnamed: 3_level_11	Unnamed: 4_level_11	Unnamed: 5_level_11
0	NaN	Graphs are temporarily unavailable due to tech...	NaN	NaN	NaN	NaN
1	1	China[B]	1270	1376	1416	NaN
2	2	India	1053	1311	1528	NaN
3	3	United States	283	322	356	NaN
4	4	Indonesia	212	258	295	NaN
5	5	Pakistan	136	208	245	NaN
6	6	Brazil	176	206	228	NaN
7	7	Nigeria	123	182	263	NaN
8	8	Bangladesh	131	161	186	NaN
9	9	Russia	146	146	149	NaN
10	10	Mexico	103	127	148	NaN
11	NaN	World total	6127	7349	8501	NaN
12	Notes: .mw-parser-output .reflist{font-size:90...	Notes: .mw-parser-output	Notes: .mw-parser-output	Notes: .mw-parser-output	Notes: .mw-parser-output	Notes: .mw-parser-output .reflist{font-size:90...

.reflist{font-size:90... .reflist{font-size:90... .reflist{font-size:90... .reflist{font-size:90...

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 [70]:
```
pd.read_html(url, match="10 most densely populated countries", flavor='bs4')[0]
```

Out[70]:

	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	

In [ ]:

In [ ]: