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' is not recognized as an internal or external command,
operable program or batch file.
Collecting lxml==4.6.4
  Downloading lxml-4.6.4-cp310-cp310-win_amd64.whl (3.5 MB)
     ---------------------------------------- 3.5/3.5 MB 2.4 MB/s eta 0:00:00
Installing collected packages: lxml
  Attempting uninstall: lxml
    Found existing installation: lxml 4.9.1
    Uninstalling lxml-4.9.1:
      Successfully uninstalled lxml-4.9.1
Successfully installed lxml-4.6.4
'mamba' is not recognized as an internal or external command,
operable program or batch file.
```

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='boldes
```

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 [6]:
```
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 [7]:
```
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 [8]:
```
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 [9]:
```
tag_object=soup.h3
tag_object
```

Out[9]:
```
<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 [10]:
```
tag_child =tag_object.b
```

```
tag_child
```

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

You can access the parent with the `parent`

In [11]:
```
parent_tag=tag_child.parent
parent_tag
```

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

this is identical to

In [12]:
```
tag_object
```

Out[12]:
```
<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> Stephen Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3><p> Salary: $73,200, 000</p></body>
```

`tag_object` sibling is the `paragraph` element

In [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 [ ]:
```

```

▶ 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 [15]:
```
tag_child['id']
```

Out[15]:
```
'boldest'
```

You can access that dictionary directly as `attrs`:

In [16]:
```
tag_child.attrs
```

Out[16]:
```
{'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 [17]:
```
tag_child.get('id')
```

Out[17]:
```
'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 [18]:
```
tag_string=tag_child.string
tag_string
```

Out[18]:
```
'Lebron James'
```

we can verify the type is Navigable String

In [19]:
```
type(tag_string)
```

Out[19]:
```
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 [20]:
```
unicode_string = str(tag_string)
unicode_string
```

Out[20]:
```
'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 [21]:
```
%%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 [22]:
```
table="<table><tr><td id='flight'>Flight No</td><td>Launch site</td> <td>Payload mass</t
```

In [23]:
```
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 [24]:
```
table_rows=table_bs.find_all('tr')
table_rows
```

Out[24]:
```
[<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 [25]:
```
first_row =table_rows[0]
first_row
```

Out[25]:
```
<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>
```

The type is `tag`

In [26]:
```
print(type(first_row))
```

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

we can obtain the child

In [27]:
```
first_row.td
```

Out[27]:
```
<td id="flight">Flight No</td>
```

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

In [28]:
```
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></t
r>
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 [29]:
```
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></t
d>
colunm 2 cell <td>80 kg</td>
```

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

In [30]:
```
list_input=table_bs .find_all(name=["tr", "td"])
list_input
```

Out[30]:
```
[<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 [31]:
```
table_bs.find_all(id="flight")
```

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

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

In [32]:
```
list_input=table_bs.find_all(href="https://en.wikipedia.org/wiki/Florida")
list_input
```

Out[32]:
```
[<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 [33]:
```
table_bs.find_all(href=True)
```

Out[33]:
```
[<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

string

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

In [34]:
```
table_bs.find_all(string="Florida")
```

Out[34]:
```
['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 [35]:
```
%%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 [36]:
```
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`

In [37]:
```
two_tables_bs= BeautifulSoup(two_tables, 'html.parser')
```

We can find the first table using the tag name table

In [38]:
```
two_tables_bs.find("table")
```

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

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

In [39]:
```
two_tables_bs.find("table",class_='pizza')
```

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

Downloading And Scraping The Contents Of A Web Page

We Download the contents of the web page:

In [40]:
```
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 [41]:
```
data  = requests.get(url).text
```

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

In [42]:
```
soup = BeautifulSoup(data,"html.parser")  # create a soup object using the variable 'data'
```

Scrape all links

In [43]:
```
for link in soup.find_all('a',href=True):  # in html anchor/link is represented
 by the tag <a>

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

```
#main-content
http://www.ibm.com
https://www.ibm.com/cloud/paks?lnk=ushpv18l1
https://www.ibm.com/security/executive-order-cybersecurity?lnk=ushpv18f1
https://www.ibm.com/consulting/technology/?lnk=ushpv18f2
https://www.ibm.com/training/credentials?lnk=ushpv18f3
https://www.ibm.com/blogs/blockchain/2021/09/dont-let-the-shipping-container-crisis-r
uin-your-holidays-this-year/?lnk=ushpv18f4
https://www.ibm.com/products/offers-and-discounts?link=ushpv18t5&lnk2=trial_mktpl_MPD
ISC
https://www.ibm.com/cloud/cloud-pak-for-automation?lnk=ushpv18t1&lnk2=trial_CloudPakA
tm&psrc=none&pexp=def
https://www.ibm.com/cloud/watson-studio?lnk=ushpv18t2&lnk2=trial_WatStudio&psrc=none&
pexp=def
https://www.ibm.com/cloud/aspera?lnk=ushpv18t3&lnk2=trial_AsperaCloud&psrc=none&pexp=
def
https://www.ibm.com/security/identity-access-management/cloud-identity?lnk=ushpv18t4&
lnk2=trial_Verify&psrc=none&pexp=def
https://www.ibm.com/search?lnk=ushpv18srch&locale=en-us&q=
https://www.ibm.com/products?lnk=ushpv18p1&lnk2=trial_mktpl&psrc=none&pexp=def
https://www.ibm.com/cloud/hybrid?lnk=ushpv18pt14
https://www.ibm.com/watson?lnk=ushpv18pt17
https://www.ibm.com/it-infrastructure?lnk=ushpv18pt19
https://www.ibm.com/us-en/products/categories?technologyTopics%5B0%5D%5B0%5D=cat.topi
c:Blockchain&isIBMOffering%5B0%5D=true&lnk=ushpv18pt4
https://www.ibm.com/us-en/products/category/technology/security?lnk=ushpv18pt9
https://www.ibm.com/us-en/products/category/technology/analytics?lnk=ushpv18pt1
https://www.ibm.com/cloud/automation?lnk=ushpv18ct21
https://www.ibm.com/quantum-computing?lnk=ushpv18pt16
https://www.ibm.com/mysupport/s/?language=en_US&lnk=ushpv18ct11
https://www.ibm.com/training/?lnk=ushpv18ct15
https://developer.ibm.com/?lnk=ushpv18ct9
https://www.ibm.com/garage?lnk=ushpv18pt18
https://www.ibm.com/docs/en?lnk=ushpv18ct14
https://www.redbooks.ibm.com/?lnk=ushpv18ct10
https://www-03.ibm.com/employment/technicaltalent/developer/?lnk=ushpv18ct2
https://www.ibm.com/case-studies/verizon-business/?lnk=ushpv18vn1
https://www.ibm.com/case-studies/verizon-business/?lnk=ushpv18vn1
https://www.ibm.com/
```

Scrape all images Tags

In [44]:
```
for link in soup.find_all('img'):# in html image is represented by the tag <img>
    print(link)
    print(link.get('src'))
```

```
<img alt="" aria-hidden="true" role="presentation" src="
2ZyB3aWR0aD0iMTA1NSIgaGVpZ2h0PSI1MjcuNSIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3Zn
IiB2ZXJzaW9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;pa
dding:0"/>
```


https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/2f/bc/20211115-ls-cloud-paks-26257-720x360.jpg

https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/80/38/20211107-26227%20X-Force-executive-order-444x320.jpg

https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/57/06/20211101-f-consulting-technology-26225.jpg

https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/fd/39/20211107-26176-credential-experience-444x320.jpg

<img alt="Demand is up but your supply isn’t?" class="ibm-resize ibm-ab-ima

ge featured-image" decoding="async" src="https://1.dam.s81c.com/public/content/dam/wo
rldwide-content/homepage/ul/g/37/67/20211107-26238-supply-chain-crisis-444x320.jpg" s
tyle="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-box;padding:
0;border:none;margin:auto;display:block;width:0;height:0;min-width:100%;max-width:10
0%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/37/67/20211
107-26238-supply-chain-crisis-444x320.jpg
<img alt="" aria-hidden="true" role="presentation" src="
2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2
ZXJzaW9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;paddin
g:0"/>

3d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="IBM Cloud Pak for Automation" class="ibm-resize ibm-ab-image trials-image"
decoding="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/hom
epage/ul/g/ab/f7/Cloud-pak-for-automation-444x260.png" style="position:absolute;top:
0;left:0;bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;dis
play:block;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:
100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/ab/f7/Cloud
-pak-for-automation-444x260.png
<img alt="IBM Cloud Pak for Automation" class="ibm-resize ibm-ab-image trials-image"
decoding="async" src="
AABAAEAAAIBRAA7" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:bo
rder-box;padding:0;border:none;margin:auto;display:block;width:0;height:0;min-width:1
00%;max-width:100%;min-height:100%;max-height:100%"/>

<img alt="" aria-hidden="true" role="presentation" src="
2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2
ZXJzaW9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;paddin
g:0"/>

3d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="IBM Watson Studio" class="ibm-resize ibm-ab-image trials-image" decoding="a
sync" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/
58/44/Watson-Studio-Desktop-21039-700x420.png" style="position:absolute;top:0;left:0;
bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:bloc
k;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/58/44/Watso
n-Studio-Desktop-21039-700x420.png
<img alt="IBM Watson Studio" class="ibm-resize ibm-ab-image trials-image" decoding="a
sync" src="
RAA7" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-box;pa
dding:0;border:none;margin:auto;display:block;width:0;height:0;min-width:100%;max-wid
th:100%;min-height:100%;max-height:100%"/>

<img alt="" aria-hidden="true" role="presentation" src="
2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2
ZXJzaW9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;paddin
g:0"/>

3d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="IBM Aspera on Cloud" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/u
l/g/e5/32/Aspera-on-Cloud-19783-700x420.png" style="position:absolute;top:0;left:0;bo
ttom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:block;
width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/e5/32/Asper

```
a-on-Cloud-19783-700x420.png
<img alt="IBM Aspera on Cloud" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="
AIBRAA7" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-bo
x;padding:0;border:none;margin:auto;display:block;width:0;height:0;min-width:100%;max
-width:100%;min-height:100%;max-height:100%"/>

<img alt="" aria-hidden="true" role="presentation" src="
2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2
ZXJzaW9uPSIxLjEiLz4="  style="max-width:100%;display:block;margin:0;border:none;paddin
g:0"/>

3d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="IBM Security Verify" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/u
l/g/15/16/cloud-transformation-trial-700x420.png" style="position:absolute;top:0;lef
t:0;bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:
block;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:10
0%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/15/16/cloud
-transformation-trial-700x420.png
<img alt="IBM Security Verify" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="
AIBRAA7" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-bo
x;padding:0;border:none;margin:auto;display:block;width:0;height:0;min-width:100%;max
-width:100%;min-height:100%;max-height:100%"/>

```

Scrape data from HTML tables

In [45]:
```
#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-DA
0321EN-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 [46]:
```
# get the contents of the webpage in text format and store in a variable called
 data
data  = requests.get(url).text
```

In [47]:
```
soup = BeautifulSoup(data,"html.parser")
```

In [48]:
```
#find a html table in the web page
table = soup.find('table') # in html table is represented by the tag <table>
```

In [49]:
```
#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
```

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

In [50]:
```
import pandas as pd
```

In [51]:
```
#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 [52]:
```
# get the contents of the webpage in text format and store in a variable called
 data
data  = requests.get(url).text
```

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

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

In [55]:

```
# we can see how many tables were found by checking the length of the tables lis
t
len(tables)
```

```
26
```

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 [56]:
```
for index,table in enumerate(tables):
    if ("10 most densely populated countries" in str(table)):
        table_index = index
print(table_index)
```

```
5
```

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

In [57]:
```
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>
 </caption>
 <tbody>
  <tr>
   <th>
    Rank
   </th>
   <th>
    Country
   </th>
   <th>
    Population
   </th>
   <th>
    Area
    <br/>
    <small>
     (km
     <sup>
      2
     </sup>
     )
    </small>
   </th>
   <th>
    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="900" dec
oding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/4/48/Fl
ag_of_Singapore.svg/23px-Flag_of_Singapore.svg.png" srcset="//upload.wikimedia.org/wi
kipedia/commons/thumb/4/48/Flag_of_Singapore.svg/35px-Flag_of_Singapore.svg.png 1.5x,
//upload.wikimedia.org/wikipedia/commons/thumb/4/48/Flag_of_Singapore.svg/45px-Flag_o
f_Singapore.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Singapore" title="Singapore">
     Singapore
    </a>
   </td>
   <td>
    5,704,000
   </td>
   <td>
    710
   </td>
   <td>
    8,033
   </td>
  </tr>
  <tr>
   <td>
    2
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="1000" de
coding="async" height="14" src="//upload.wikimedia.org/wikipedia/commons/thumb/f/f9/F
lag_of_Bangladesh.svg/23px-Flag_of_Bangladesh.svg.png" srcset="//upload.wikimedia.or
g/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/Flag_of_Bangladesh.svg/46px
-Flag_of_Bangladesh.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Bangladesh" title="Bangladesh">
     Bangladesh
    </a>
   </td>
   <td>
    171,670,000
   </td>
   <td>
    143,998
   </td>
   <td>
    1,192
   </td>
```

```
  </tr>
  <tr>
   <td>
    3
   </td>
   <td align="left">
    <p>
     <span class="flagicon">
      <img alt="" class="thumbborder" data-file-height="600" data-file-width="1200" d
ecoding="async" height="12" src="//upload.wikimedia.org/wikipedia/commons/thumb/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_Palestine.svg.png 1.5
x, //upload.wikimedia.org/wikipedia/commons/thumb/0/00/Flag_of_Palestine.svg/46px-Fla
g_of_Palestine.svg.png 2x" width="23"/>
     </span>
     <a href="/wiki/State_of_Palestine" title="State of Palestine">
      Palestine
     </a>
    </p>
   </td>
   <td>
    5,266,785
   </td>
   <td>
    6,020
   </td>
   <td>
    847
   </td>
  </tr>
  <tr>
   <td>
    4
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" dec
oding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/59/Fl
ag_of_Lebanon.svg/23px-Flag_of_Lebanon.svg.png" srcset="//upload.wikimedia.org/wikipe
dia/commons/thumb/5/59/Flag_of_Lebanon.svg/35px-Flag_of_Lebanon.svg.png 1.5x, //uploa
d.wikimedia.org/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/45px-Flag_of_Lebano
n.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Lebanon" title="Lebanon">
     Lebanon
    </a>
   </td>
   <td>
    6,856,000
   </td>
   <td>
    10,452
   </td>
   <td>
    656
   </td>
  </tr>
  <tr>
```

```
   <td>
    5
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" dec
oding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Fl
ag_of_the_Republic_of_China.svg/23px-Flag_of_the_Republic_of_China.svg.png" srcset
="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.s
vg/35px-Flag_of_the_Republic_of_China.svg.png 1.5x, //upload.wikimedia.org/wikipedia/
commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/45px-Flag_of_the_Republic_of_Chi
na.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Taiwan" title="Taiwan">
     Taiwan
    </a>
   </td>
   <td>
    23,604,000
   </td>
   <td>
    36,193
   </td>
   <td>
    652
   </td>
  </tr>
  <tr>
   <td>
    6
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" dec
oding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/0/09/Fl
ag_of_South_Korea.svg/23px-Flag_of_South_Korea.svg.png" srcset="//upload.wikimedia.or
g/wikipedia/commons/thumb/0/09/Flag_of_South_Korea.svg/35px-Flag_of_South_Korea.svg.p
ng 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/0/09/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>
    51,781,000
   </td>
   <td>
    99,538
   </td>
   <td>
    520
   </td>
  </tr>
  <tr>
   <td>
    7
   </td>
```

<td align="left">

Rwanda

</td>
<td>
12,374,000
</td>
<td>
26,338
</td>
<td>
470
</td>
</tr>
<tr>
<td>
8
</td>
<td align="left">

Haiti

</td>
<td>
11,578,000
</td>
<td>
27,065
</td>
<td>
428
</td>
</tr>
<tr>
<td>
9
</td>
<td align="left">

```
ag_of_the_Netherlands.svg/23px-Flag_of_the_Netherlands.svg.png" srcset="//upload.wiki
media.org/wikipedia/commons/thumb/2/20/Flag_of_the_Netherlands.svg/35px-Flag_of_the_N
etherlands.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/2/20/Flag_of_
the_Netherlands.svg/45px-Flag_of_the_Netherlands.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Netherlands" title="Netherlands">
     Netherlands
    </a>
   </td>
   <td>
    17,660,000
   </td>
   <td>
    41,526
   </td>
   <td>
    425
   </td>
  </tr>
  <tr>
   <td>
    10
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="800" data-file-width="1100" de
coding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/d/d4/F
lag_of_Israel.svg/21px-Flag_of_Israel.svg.png" srcset="//upload.wikimedia.org/wikiped
ia/commons/thumb/d/d4/Flag_of_Israel.svg/32px-Flag_of_Israel.svg.png 1.5x, //upload.w
ikimedia.org/wikipedia/commons/thumb/d/d4/Flag_of_Israel.svg/41px-Flag_of_Israel.svg.
png 2x" width="21"/>
    </span>
    <a href="/wiki/Israel" title="Israel">
     Israel
    </a>
   </td>
   <td>
    9,430,000
   </td>
   <td>
    22,072
   </td>
   <td>
    427
   </td>
  </tr>
 </tbody>
</table>
```


In [58]:

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

for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col != []):
        rank = col[0].text
        country = col[1].text
        population = col[2].text.strip()
```

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

population_data
```

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

	Rank	Country	Population	Area	Density
0	1	Singapore	5,704,000	710	8,033
1	2	Bangladesh	171,670,000	143,998	1,192
2	3	\n Palestine\n\n	5,266,785	6,020	847
3	4	Lebanon	6,856,000	10,452	656
4	5	Taiwan	23,604,000	36,193	652
5	6	South Korea	51,781,000	99,538	520
6	7	Rwanda	12,374,000	26,338	470
7	8	Haiti	11,578,000	27,065	428
8	9	Netherlands	17,660,000	41,526	425
9	10	Israel	9,430,000	22,072	427

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 [59]:
```
pd.read_html(str(tables[5]), flavor='bs4')
```

Out[59]:
```
[   Rank       Country  Population  Area(km2)  Density(pop/km2)
 0     1     Singapore     5704000        710              8033
 1     2    Bangladesh   171670000     143998              1192
 2     3     Palestine     5266785       6020               847
 3     4       Lebanon     6856000      10452               656
 4     5        Taiwan    23604000      36193               652
 5     6   South Korea    51781000      99538               520
 6     7        Rwanda    12374000      26338               470
 7     8         Haiti    11578000      27065               428
 8     9   Netherlands    17660000      41526               425
 9    10        Israel     9430000      22072               427]
```

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

list.

In [60]:
```
population_data_read_html = pd.read_html(str(tables[5]), flavor='bs4')[0]

population_data_read_html
```

Out[60]:

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5704000	710	8033
1	2	Bangladesh	171670000	143998	1192
2	3	Palestine	5266785	6020	847
3	4	Lebanon	6856000	10452	656
4	5	Taiwan	23604000	36193	652
5	6	South Korea	51781000	99538	520
6	7	Rwanda	12374000	26338	470
7	8	Haiti	11578000	27065	428
8	9	Netherlands	17660000	41526	425
9	10	Israel	9430000	22072	427

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 [61]:
```
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 [62]:
```
len(dataframe_list)
```

Out[62]:
```
26
```

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

In [63]:
```
dataframe_list[5]
```

Out[63]:

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5704000	710	8033
1	2	Bangladesh	171670000	143998	1192
2	3	Palestine	5266785	6020	847
3	4	Lebanon	6856000	10452	656
4	5	Taiwan	23604000	36193	652
5	6	South Korea	51781000	99538	520
6	7	Rwanda	12374000	26338	470
7	8	Haiti	11578000	27065	428
8	9	Netherlands	17660000	41526	425

9	10	Israel	9430000	22072	427

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 [64]:
```
pd.read_html(url, match="10 most densely populated countries", flavor='bs4')[0]
```

Out[64]:

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5704000	710	8033
1	2	Bangladesh	171670000	143998	1192
2	3	Palestine	5266785	6020	847
3	4	Lebanon	6856000	10452	656
4	5	Taiwan	23604000	36193	652
5	6	South Korea	51781000	99538	520
6	7	Rwanda	12374000	26338	470
7	8	Haiti	11578000	27065	428
8	9	Netherlands	17660000	41526	425
9	10	Israel	9430000	22072	427

In [ ]:

In [ ]: