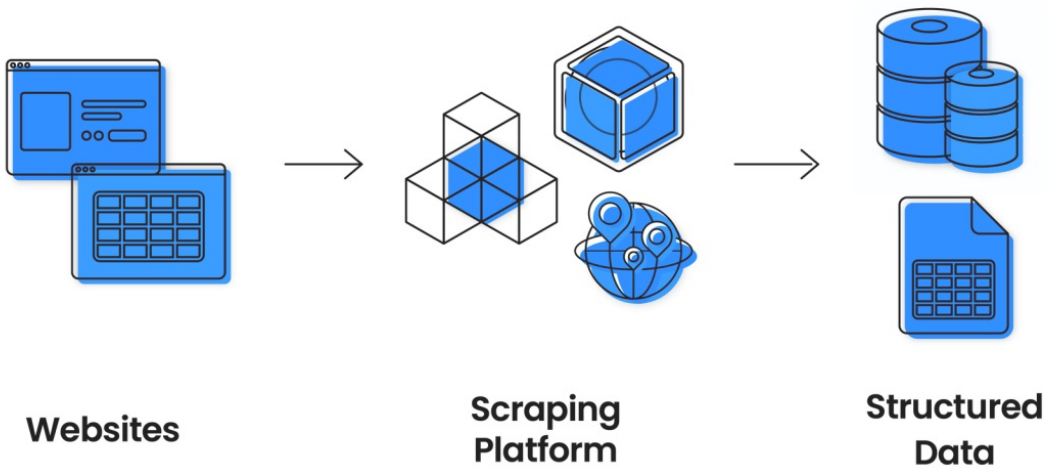
Web Scrapping with BeautifulSoup Python

Web scraping is a powerful technique used to extract data from websites. In this notebook, we explore how to use the BeautifulSoup library in Python to parse HTML content and retrieve structured information. Whether you're collecting data for research, analytics, or automation, BeautifulSoup makes it easy to navigate and extract data from complex webpages.



Key Points Covered: ✅ What is Web Scraping? An introduction to extracting structured data from websites.

✅ Introduction to BeautifulSoup: A powerful Python library used for parsing HTML and XML documents.

✅ Installation Requirements: Required libraries such as beautifulsoup4 and requests.

✅ Fetching HTML Content: Using the requests module to get raw HTML from a webpage.

✅ Parsing the HTML: Creating a BeautifulSoup object to parse and traverse HTML content.

✅ Using Tags and Attributes: How to find elements using tags and attributes like href.

✅ Extracting Text Content: Getting clean text from HTML tags using .text or .get_text() methods.

✅ Handling Tables: Navigating HTML tables to extract structured data such as population, rank, and area.

✅ Storing in DataFrame: Collecting the scraped data and organizing it in a Pandas DataFrame for further analysis.

Installing Requirements

Before we begin scraping data, it's important to set up the necessary tools. In this section, we install the key Python libraries that make web scraping simple and efficient. The two main libraries we'll use are requests for fetching web pages and beautifulsoup4 for parsing HTML content. Installing these packages ensures your environment is ready for scraping tasks.

In [5]:
```
!pip install bs4
!pip install lxml
!pip install html5lib
!pip install requests
```

```
Requirement already satisfied: bs4 in c:\users\tarun\anaconda3\lib\site-packages (0.0.2)
Requirement already satisfied: beautifulsoup4 in c:\users\tarun\anaconda3\lib\site-packages (from bs4) (4.12.3)
Requirement already satisfied: soupsieve>1.2 in c:\users\tarun\anaconda3\lib\site-packages (from beautifulsoup4-
>bs4) (2.5)
Requirement already satisfied: lxml in c:\users\tarun\anaconda3\lib\site-packages (5.3.0)
Requirement already satisfied: html5lib in c:\users\tarun\anaconda3\lib\site-packages (1.1)
Requirement already satisfied: six>=1.9 in c:\users\tarun\anaconda3\lib\site-packages (from html5lib) (1.17.0)
Requirement already satisfied: webencodings in c:\users\tarun\anaconda3\lib\site-packages (from html5lib) (0.5.1
)
Requirement already satisfied: requests in c:\users\tarun\anaconda3\lib\site-packages (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\tarun\anaconda3\lib\site-packages (from requ
ests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\tarun\anaconda3\lib\site-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\tarun\anaconda3\lib\site-packages (from requests)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\tarun\anaconda3\lib\site-packages (from requests)
(2025.6.15)
```

Importing BeautifulSoup Python Library

Before we begin scraping data from the web, we need to import the necessary tools. BeautifulSoup, part of the bs4 module, is a powerful Python library used for parsing HTML and XML documents. It allows us to navigate the structure of web pages and extract the data we need efficiently. Along with BeautifulSoup, we often use the requests library to fetch the content of webpages.

In [7]:
```
from bs4 import BeautifulSoup
import requests
```

In [25]:
```
%%html
<!DOCTYPE html>
<html>
<head>
<title>Salary Page</title>
</head>
<body>
<h3><b id='boldest'> Harry Potter </b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Jack Sparrow </h3>
<p> Salary: $85,000, 000 </p>
<h3> Tony Stark </h3>
<p> Salary: $173,200, 000</p>
</body>
</html>
```

Harry Potter

Salary: $ 92,000,000

Jack Sparrow

Salary: $85,000, 000

Tony Stark

Salary: $173,200, 000

In [26]:
```
html="<!DOCTYPE html><html><head><title> Salary Page </title></head><body><h3><b id='boldest'> Harry Potter </b:
```

Once we have the raw HTML content of a webpage, the next step is to parse it using BeautifulSoup. The line soup = BeautifulSoup(html, 'html.parser') creates a BeautifulSoup object, which acts like a structured representation of the HTML document. Using the 'html.parser' tells BeautifulSoup to use Python's built-in HTML parser to interpret the document. This object (soup) allows us to easily search, navigate, and manipulate elements within the HTML content.

In [27]:
```
soup = BeautifulSoup(html,'html.parser')
```

The print(soup.prettify()) command displays the HTML content in a well-formatted, indented structure, making it easier to read and understand the page's layout.

In [28]:
```
print(soup.prettify())
```

```
<!DOCTYPE html>
<html>
 <head>
  <title>
   Salary Page
  </title>
 </head>
 <body>
  <h3>
   <b id="boldest">
    Harry Potter
   </b>
  </h3>
  <p>
   Salary: $ 92,000,000
  </p>
  <h3>
   Jack Sparrow
  </h3>
  <p>
   Salary: $85,000,000
  </p>
  <h3>
   Tony Stark
  </h3>
  <p>
   Salary: $173,200,000
  </p>
 </body>
</html>
```

In BeautifulSoup, Tag objects represent individual HTML elements, such as h1,h2, a, div, p, etc. Each tag in an HTML document is converted into a Tag object, which allows you to access its content, attributes, and even its nested elements. For example, using soup.title returns the title tag as a Tag object, from which you can extract the tag name, text, or attributes. Tag objects are central to navigating and manipulating the structure of an HTML page using BeautifulSoup.

In [29]:
```
tag_object=soup.title
print('tag_object', tag_object)
```

```
tag_object <title> Salary Page </title>
```

In [30]:
```
print('tag object type', type(tag_object))
```

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

In [34]:
```
tag_object =soup.h3
tag_object
```

Out[34]:
```
 <h3><b id="boldest"> Harry Potter </b></h3>
```

In [38]:
```
tag_child=tag_object.b
tag_child
```

Out[38]:
```
 <b id="boldest"> Harry Potter </b>
```

In [39]:
```
parent_tag = tag_child.parent
parent_tag
```

Out[39]:
```
 <h3><b id="boldest"> Harry Potter </b></h3>
```

In [40]:
```
tag_object
```

Out[40]:
```
 <h3><b id="boldest"> Harry Potter </b></h3>
```

In [41]:
```
tag_object.parent
```

Out[41]:
```
 <body><h3><b id="boldest"> Harry Potter </b></h3><p> Salary: $ 92,000,000 </p><h3> Jack Sparrow </h3><p> Salary
 : $85,000,000 </p><h3> Tony Stark </h3><p> Salary: $173,200,000</p></body>
```

Parent, Children, and Sibling in BeautifulSoup

In BeautifulSoup, you can easily navigate the HTML structure using relationships. The .parent gives the tag that directly contains another tag. .children returns all tags nested inside a tag. .next_sibling and .previous_sibling let you move to elements that are at the same level in the HTML. These tools help you explore and extract data from the webpage efficiently.

In [42]:
```
sibling_1=tag_object.next_sibling
sibling_1
```

Out[42]:
```
<p> Salary: $ 92,000,000 </p>
```

In [43]:
```
sibling_2=sibling_1.next_sibling
sibling_2
```

Out[43]:
```
<h3> Jack Sparrow </h3>
```

In [45]:
```
sibling_3=sibling_2.next_sibling
sibling_3
```

Out[45]:
```
<p> Salary: $85,000,000 </p>
```

In [46]:
```
tag_child['id']
```

Out[46]:
```
'boldest'
```

In [50]:
```
tag_string=sibling_3.string
tag_string
```

Out[50]:
```
' Salary: $85,000,000 '
```

In [54]:
```
unicode_string = str(sibling_3)
unicode_string
```

Out[54]:
```
'<p> Salary: $85,000,000 </p>'
```

Filter

Filtering in BeautifulSoup allows you to search for specific HTML elements based on tag names, attributes, text, or custom functions. You can use filters with methods like .find(), .find_all() to locate only the elements that match certain criteria. For example, you can filter all (a) tags with a specific class or all tags containing a certain word. This makes it easy to target and extract exactly the content you need from complex web pages.

In [55]:
```
%%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

In [56]:
```
table="<table><tr><td id='flight' >Flight No</td><td>Launch site</td><td>Payload mass</td></tr><tr><td>1</td><t
```

In [57]:
```
table_bs = BeautifulSoup(table, "html.parser")
```

In [59]:
```
table_rows=table_bs.find_all('tr')
table_rows
```

Out[59]:
```
[<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>]
```

In [70]:
```
first_row=table_rows[0]
first_row
```

Out[70]:
```
<tr><td id="flight">Flight No</td><td>Launch site</td><td>Payload mass</td></tr>
```

In [61]:
```
print(type(first_row))
```

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

In [71]:
```
first_row.td
```

Out[71]:
```
<td id="flight">Flight No</td>
```

In [74]:
```
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>
```

In [75]:
```
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>
```

In [77]:
```
list_input=table_bs.find_all(name=["tr", "td"])
list_input
```

Out[77]:
```
[<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>]
```

In [78]:
```
table_bs.find_all(href=True)
```

Out[78]:
```
[<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>]
```

In [83]:
```
soup.find_all(id='boldest')
```

Out[83]:
```
[<b id="boldest"> Harry Potter </b>]
```

Find()

The find() method in BeautifulSoup is used to locate the first matching HTML element based on tag name, attributes, or other filters. It returns a single Tag object, making it ideal when you only need one element—like the first div,a, or any specific tag with a certain class or ID. This method helps narrow down your search quickly and efficiently when parsing web pages.

In [84]:
```
%%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

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

In [85]:
```
two_tables="<h3>Rocket Launch </h3><p><table class='rocket'><tr><td>Flight No</td><td>Launch site</td> <td>Payl
```

In [86]:
```
two_tables_bs= BeautifulSoup(two_tables, 'html.parser')
```

In [89]:
```
two_tables_bs.find('table')
```

Out[89]:
```
<table class="rocket"><tr><td>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr><td>1</td><td>Fl
orida</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>
```

In [90]:
```
two_tables_bs.find("table",class_='pizza')
```

Out[90]:
```
<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

To begin web scraping, the first step is to download the HTML content of the target webpage. This is typically done using the requests library in Python, which sends an HTTP request to the webpage's URL and retrieves its source code. Once the content is downloaded, we use BeautifulSoup to parse and extract meaningful data from the raw HTML. This process forms the foundation for scraping structured data such as tables, links, and text from any website.

In [91]:
```
url = "http://www.ibm.com"
```

In [92]:
```
data  = requests.get(url).text
```

In [93]:
```
soup = BeautifulSoup(data,"html.parser")
```

In [94]:
```
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/sports/wimbledon?lnk=hpls1uk
https://www.ibm.com/sports/wimbledon?lnk=hpls1uk
https://www.ibm.com/products/watsonx-orchestrate?lnk=hpls2in
https://www.ibm.com/downloads/documents/us-en/10a99803cbafdc57
https://www.ibm.com/in-en/campaign/ai-agents-incubation-workshop#urx
https://www.ibm.com/store/en/in/products/EIDS2PRM
https://www.ibm.com/case-studies/tcs?lnk=hpinrc2
https://www.ibm.com/products/power?lnk=hpls4in
https://www.ibm.com/granite?lnk=dev
https://developer.ibm.com/technologies/artificial-intelligence?lnk=dev
https://www.ibm.com/products/watsonx-code-assistant?lnk=dev
https://www.ibm.com/products/instana?lnk=dev
https://www.ibm.com/thought-leadership/institute-business-value/report/ceo-generative-ai?lnk=bus
https://www.ibm.com/think/videos/ai-academy
https://www.ibm.com/products/watsonx-orchestrate/ai-agent-for-hr?lnk=bus
https://www.ibm.com/products/guardium-data-security-center?lnk=bus
https://www.ibm.com/new/announcements/ibm-named-leader-in-2025-gartner-magic-quadrant-for-observability-platform
s
https://www.ibm.com/in-en/artificial-intelligence?lnk=ProdC
https://www.ibm.com/hybrid-cloud?lnk=ProdC
https://www.ibm.com/consulting?lnk=ProdC
https://www.ibm.com/artificial-intelligence?lnk=ProdC
https://www.ibm.com/granite?lnk=ProdC
https://www.ibm.com/consulting?lnk=ProdC
https://www.ibm.com/analytics?lnk=ProdC
https://www.ibm.com/aiops?lnk=ProdC
https://www.ibm.com/events/reg/flow/ibm/wlh63kmb/landing/page/landing?utm_source=ibmcomutm_id=ibmcom
https://www.ibm.com/servers?lnk=ProdC
https://www.ibm.com/database?lnk=ProdC
https://www.ibm.com/security?lnk=ProdC
https://www.ibm.com/in-en/about?lnk=inside
https://www.ibm.com/history?lnk=inside
https://research.ibm.com?lnk=inside
https://www.ibm.com/quantum?lnk=inside
https://www.ibm.com/in-en/careers?lnk=hpii1in
https://skillsbuild.org?lnk=hpii1au
```

Scrap all images from the link

Scraping images involves finding all (img) tags in the HTML and extracting their src attributes, which contain the image URLs. Using BeautifulSoup, we can easily loop through these tags and download each image using the requests library. This technique is useful for collecting image datasets or saving media content from websites.

In [97]:
```
for link in soup.find_all('img'):# in html image is represented by the tag <img>
    print(link)
    print(link.get('src'))
```

```
<img alt="https://1.dam.s81c.com/m/77089e2b3c31c4d4/original/11-best-practices-MDM-FR.jpg" class="cmp-image__ima
ge" height="260" itemprop="contentUrl" loading="lazy" src="https://assets.ibm.com/is/image/ibm/31747827-7435-4b3
a-a708e3121f4096c4?ts=1752495085444&amp;dpr=off" srcset="
AABAAEAAAICTAEAOw==" width="600"/>
https://assets.ibm.com/is/image/ibm/31747827-7435-4b3a-a708e3121f4096c4?ts=1752495085444&dpr=off
<img alt="Instana Community Open House" class="cmp-image__image" height="284" itemprop="contentUrl" loading="laz
y" src="https://assets.ibm.com/is/image/ibm/Instana_Community_Open_House?ts=1744134177598&amp;dpr=off" srcset="d
ata:image/gif;base64,R0lGODlhAQABAAAAACH5BAEKAAEALAAAAAABAAEAAAICTAEAOw==" width="284"/>
https://assets.ibm.com/is/image/ibm/Instana_Community_Open_House?ts=1744134177598&dpr=off
<img alt="Thumbnail for the AI Productivity Sales thought leadership" class="cmp-image__image" height="320" item
prop="contentUrl" loading="lazy" src="https://assets.ibm.com/is/image/ibm/1573-ai-productivity-sales_568x320?ts=
1752154279407&amp;dpr=off" srcset="" w
idth="568"/>
https://assets.ibm.com/is/image/ibm/1573-ai-productivity-sales_568x320?ts=1752154279407&dpr=off
<img alt="Introducing IBM Power11: designed to be the most resilient server in the history of the platform." cla
ss="cmp-image__image" height="1350" itemprop="contentUrl" loading="lazy" src="https://assets.ibm.com/is/image/ib
m/ibm-power11-three-servers?ts=1752495674950&amp;dpr=off" srcset="
AAEALAAAAAABAAEAAAICTAEAOw==" width="1080"/>
https://assets.ibm.com/is/image/ibm/ibm-power11-three-servers?ts=1752495674950&dpr=off
<img alt="Still image derived from the Automation toolkit representing IT infrastructure with a scanner that ide
ntifies problematic issues across the application stacks." class="cmp-image__image" height="140" itemprop="conte
ntUrl" loading="lazy" src="https://assets.ibm.com/is/image/ibm/instana-gartner-magic-quadrant-promo-banner-406x1
40?ts=1752756881184&amp;dpr=off" srcset="
w==" width="406"/>
https://assets.ibm.com/is/image/ibm/instana-gartner-magic-quadrant-promo-banner-406x140?ts=1752756881184&dpr=off
```

Scrap all tables

To scrape all images from a webpage, we search for all img tags using BeautifulSoup, as these tags contain image data. The src attribute of each img tag holds the image URL. By extracting these URLs and optionally downloading them using the requests library, we can collect all the images from the webpage. This technique is especially useful for building image datasets or saving visual content from a site.

In [98]:
```
#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/datas
```

In [99]:
```
# get the contents of the webpage in text format and store in a variable called data
data  = requests.get(url).text
```

In [100…
```
soup = BeautifulSoup(data,"html.parser")
```

In [101…
```
#find a html table in the web page
table = soup.find('table') # in html table is represented by the tag <table>
```

In [102…
```
#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

HTML tables often contain valuable structured data, and with the help of BeautifulSoup and Pandas, we can easily extract this data into a usable format. By locating the (`<table>`) tag and navigating through its rows (`<tr>`) and columns (`<td>` or `<th>`), we can collect the content into a list of dictionaries or lists. Pandas then allows us to convert this extracted data into a DataFrame, making it easy to clean, analyze, and visualize. This is particularly useful for scraping tables from websites such as Wikipedia, government portals, or any site with tabular data.

In [103…
```
import pandas as pd

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

In [104…
```
data  = requests.get(url).text
```

In [123…
```
soup = BeautifulSoup(data,"html.parser")
```

In [124…
```
tables = soup.find_all('table')
```

In [107…
```
len(tables)
```

Out[107… 26

In [108…
```
for index,table in enumerate(tables):
    if ("10 most densely populated countries" in str(table)):
        table_index = index
print(table_index)
```

5

In [112…
```
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">
    <span class="cite-bracket">
     [
    </span>
    101
```

```
    <span class="cite-bracket">
     ]
    </span>
   </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="600" data-file-width="900" decoding="async" height=
"15" src="//upload.wikimedia.org/wikipedia/commons/thumb/4/48/Flag_of_Singapore.svg/40px-Flag_of_Singapore.svg.p
ng" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/4/48/Flag_of_Singapore.svg/60px-Flag_of_Singapore.svg
.png 2x" width="23"/>
      </span>
     </span>
    </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="307" data-file-width="512" decoding="async" height=
"14" src="//upload.wikimedia.org/wikipedia/commons/thumb/f/f9/Flag_of_Bangladesh.svg/40px-Flag_of_Bangladesh.svg
.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/f/f9/Flag_of_Bangladesh.svg/60px-Flag_of_Bangladesh
.svg.png 2x" width="23"/>
      </span>
     </span>
```

```
    </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 nowrap">
      <span class="mw-image-border" typeof="mw:File">
       <span>
        <img alt="" class="mw-file-element" data-file-height="600" data-file-width="1200" decoding="async" heigh
t="12" src="//upload.wikimedia.org/wikipedia/commons/thumb/0/00/Flag_of_Palestine.svg/40px-Flag_of_Palestine.svg
.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/0/00/Flag_of_Palestine.svg/60px-Flag_of_Palestine.s
vg.png 2x" width="23"/>
       </span>
      </span>
     </span>
     <a href="/wiki/Palestine" title="Palestine">
      Palestine
     </a>
     <sup class="reference" id="cite_ref-107">
      <a href="#cite_note-107">
       <span class="cite-bracket">
        [
       </span>
       note 3
       <span class="cite-bracket">
        ]
       </span>
      </a>
     </sup>
     <sup class="reference" id="cite_ref-108">
      <a href="#cite_note-108">
       <span class="cite-bracket">
        [
       </span>
       102
       <span class="cite-bracket">
        ]
       </span>
      </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="600" data-file-width="900" decoding="async" height=
"15" src="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/40px-Flag_of_the
_Republic_of_China.svg.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_
China.svg/60px-Flag_of_the_Republic_of_China.svg.png 2x" width="23"/>
      </span>
     </span>
    </span>
```

```
    <a href="/wiki/Taiwan" title="Taiwan">
     Taiwan
    </a>
    <sup class="reference" id="cite_ref-109">
     <a href="#cite_note-109">
      <span class="cite-bracket">
       [
      </span>
      note 4
      <span class="cite-bracket">
       ]
      </span>
     </a>
    </sup>
   </td>
   <td>
    23,580,712
   </td>
   <td>
    35,980
   </td>
   <td>
    655
   </td>
  </tr>
  <tr>
   <td>
    5
   </td>
   <td align="left">
    <span class="flagicon nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="600" data-file-width="900" decoding="async" height=
"15" src="//upload.wikimedia.org/wikipedia/commons/thumb/0/09/Flag_of_South_Korea.svg/40px-Flag_of_South_Korea.s
vg.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/0/09/Flag_of_South_Korea.svg/60px-Flag_of_South_K
orea.svg.png 2x" width="23"/>
      </span>
     </span>
    </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="1920" data-file-width="2880" decoding="async" heigh
t="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/40px-Flag_of_Lebanon.svg.png
" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/60px-Flag_of_Lebanon.svg.png 2
x" width="23"/>
      </span>
     </span>
    </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="600" data-file-width="900" decoding="async" height=
"15" src="//upload.wikimedia.org/wikipedia/commons/thumb/1/17/Flag_of_Rwanda.svg/40px-Flag_of_Rwanda.svg.png" sr
cset="//upload.wikimedia.org/wikipedia/commons/thumb/1/17/Flag_of_Rwanda.svg/60px-Flag_of_Rwanda.svg.png 2x" wid
th="23"/>
      </span>
     </span>
    </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="600" data-file-width="1000" decoding="async" height
="14" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/50/Flag_of_Burundi.svg/40px-Flag_of_Burundi.svg.png"
srcset="//upload.wikimedia.org/wikipedia/commons/thumb/5/50/Flag_of_Burundi.svg/60px-Flag_of_Burundi.svg.png 2x"
width="23"/>
      </span>
     </span>
    </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 nowrap">
     <span class="mw-image-border" typeof="mw:File">
      <span>
       <img alt="" class="mw-file-element" data-file-height="800" data-file-width="1100" decoding="async" height
="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Flag_of_Israel.svg/40px-Flag_of_Israel.svg.png" s
rcset="//upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Flag_of_Israel.svg/60px-Flag_of_Israel.svg.png 2x" wi
dth="21"/>
      </span>
     </span>
    </span>
    <a href="/wiki/Israel" title="Israel">
     Israel
    </a>
   </td>
   <td>
    9,402,617
   </td>
   <td>
    21,937
   </td>
```

```
  <td>
   429
  </td>
 </tr>
 <tr>
  <td>
   10
  </td>
  <td align="left">
   <span class="flagicon nowrap">
    <span class="mw-image-border" typeof="mw:File">
     <span>
      <img alt="" class="mw-file-element" data-file-height="600" data-file-width="900" decoding="async" height=
"15" src="//upload.wikimedia.org/wikipedia/en/thumb/4/41/Flag_of_India.svg/40px-Flag_of_India.svg.png" srcset="/
/upload.wikimedia.org/wikipedia/en/thumb/4/41/Flag_of_India.svg/60px-Flag_of_India.svg.png 2x" width="23"/>
     </span>
    </span>
   </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>
</tbody>
</table>
```


In [116…

```python
import pandas as pd

# Create an empty list to store rows
data_rows = []

# Iterate through each row in the HTML table
for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if col != []:
        rank = col[0].text.strip()
        country = col[1].text.strip()
        population = col[2].text.strip()
        area = col[3].text.strip()
        density = col[4].text.strip()

        # Append dictionary to list
        data_rows.append({
            "Rank": rank,
            "Country": country,
            "Population": population,
            "Area": area,
            "Density": density
        })

# Convert list of dicts to DataFrame once at the end
population_data = pd.DataFrame(data_rows)

population_data
```

Out[116…

	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	Palestine[note 3][102]	5,223,000	6,025	867
3	4	Taiwan[note 4]	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	Israel	9,402,617	21,937	429
9	10	India	1,389,637,446	3,287,263	423

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

In [117…
```
pd.read_html(str(tables[5]), flavor='bs4')
```

```
C:\Users\tarun\AppData\Local\Temp\ipykernel_27096\4059676793.py:1: FutureWarning: Passing literal html to 'read_
html' is deprecated and will be removed in a future version. To read from a literal string, wrap it in a 'String
IO' object.
  pd.read_html(str(tables[5]), flavor='bs4')
```

Out[117…
```
[   Rank                Country  Population  Area (km2)  Density (pop/km2)
 0     1              Singapore     5921231         719               8235
 1     2             Bangladesh   165650475      148460               1116
 2     3  Palestine[note 3][102]     5223000        6025                867
 3     4          Taiwan[note 4]    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                  Israel     9402617       21937                429
 9    10                   India  1389637446     3287263                423]
```

In [118…
```
population_data_read_html = pd.read_html(str(tables[5]), flavor='bs4')[0]

population_data_read_html
```

```
C:\Users\tarun\AppData\Local\Temp\ipykernel_27096\3554864673.py:1: FutureWarning: Passing literal html to 'read_
html' is deprecated and will be removed in a future version. To read from a literal string, wrap it in a 'String
IO' object.
  population_data_read_html = pd.read_html(str(tables[5]), flavor='bs4')[0]
```

Out[118…

	Rank	Country	Population	Area (km2)	Density (pop/km2)
0	1	Singapore	5921231	719	8235
1	2	Bangladesh	165650475	148460	1116
2	3	Palestine[note 3][102]	5223000	6025	867
3	4	Taiwan[note 4]	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	Israel	9402617	21937	429
9	10	India	1389637446	3287263	423

Scrape data from HTML tables into a DataFrame using read_html

Pandas provides a powerful method called read_html() that allows you to directly extract tables from an HTML page into a DataFrame with minimal code. It automatically detects (`<table>`) tags and converts them into one or more DataFrames. This approach is quick, efficient, and works well for clean and properly formatted tables, especially on sites like Wikipedia. All you need to provide is the page URL or the HTML content, and Pandas handles the rest.

In [119…
```
dataframe_list = pd.read_html(url, flavor='bs4')
```

In [120…
```
len(dataframe_list)
```

Out[120… 26

In [121…
```
dataframe_list[5]
```

Out[121…

	Rank	Country	Population	Area (km2)	Density (pop/km2)
0	1	Singapore	5921231	719	8235
1	2	Bangladesh	165650475	148460	1116
2	3	Palestine[note 3][102]	5223000	6025	867
3	4	Taiwan[note 4]	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	Israel	9402617	21937	429
9	10	India	1389637446	3287263	423

In [122…

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

Out[122…

	Rank	Country	Population	Area (km2)	Density (pop/km2)
0	1	Singapore	5921231	719	8235
1	2	Bangladesh	165650475	148460	1116
2	3	Palestine[note 3][102]	5223000	6025	867
3	4	Taiwan[note 4]	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	Israel	9402617	21937	429
9	10	India	1389637446	3287263	423

Conclusion: Mastering Web Scraping with BeautifulSoup and Pandas

In this notebook, we explored the complete process of web scraping using Python's BeautifulSoup and Pandas libraries. Starting from importing libraries and downloading web pages, we learned how to parse HTML using BeautifulSoup, navigate through tag structures, and handle relationships like parent, children, and siblings. We covered how to filter elements, scrape tables, and extract useful data such as images and structured tables. By the end, we saw how to convert scraped HTML tables directly into Pandas DataFrames using both BeautifulSoup and the powerful read_html() method. These tools together make it easier to collect and organize data from websites for analysis, automation, or research purposes.