😊 Web Scraping in Python

What is Web Scraping?

Web scraping is the process of extracting data from websites and converting it into a structured format for analysis or storage. Python provides several libraries like **BeautifulSoup, Scrapy, and Selenium** to automate this process. Web scraping allows users to collect publicly available data efficiently without manual copying.

Use Cases of Web Scraping in Python

1. **Price Monitoring and Comparison** – Businesses track competitor pricing for dynamic pricing strategies.
2. **Market Research and Data Collection** – Researchers collect large amounts of online data for analysis.
3. **Lead Generation** – Companies scrape business directories and social media for contact information.
4. **News Aggregation** – Automating the collection of news articles from multiple sources.
5. **SEO and Web Analytics** – Extracting search engine rankings, keyword trends, and competitor data.
6. **Stock Market and Financial Data** – Fetching live or historical financial data for investment analysis.
7. **E-commerce Product Scraping** – Gathering product details, reviews, and ratings for trend analysis.

Benefits of Web Scraping in Python

- **Automation** – Eliminates the need for manual data extraction.
- **Efficiency** – Extracts large volumes of data quickly.
- **Cost-Effective** – Reduces the cost of data collection compared to hiring manual workers.
- **Real-time Updates** – Enables tracking of real-time data for informed decision-making.
- **Scalability** – Easily handles scraping of thousands or millions of web pages.

Python's web scraping capabilities make it a powerful tool for businesses, researchers, and analysts looking to extract valuable insights from online sources.

To begin with the Lab

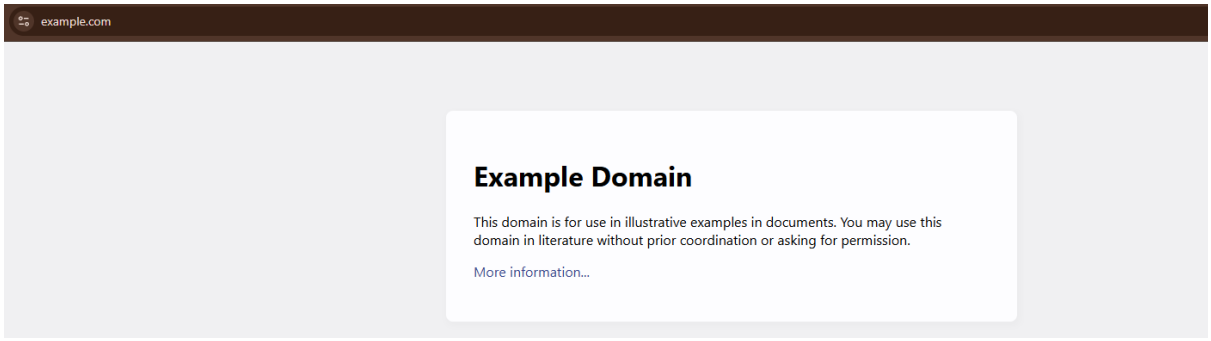
1. Before starting this, you should have a prior knowledge of basic HTML and CSS.
2. Now you need to install some libraries to work in this lab.

```
conda install requests
conda install lxml
conda install bs4
```

if you are not using the Anaconda Installation, you can use **pip install** instead of **conda install**, for example:

```
pip install requests
pip install lxml
pip install bs4
```

3. In a new tab, open example.com and right click anywhere on the page to open the menu, and from that choose view page source.



4. You will be able to see the basic HTML code for the page. Now we have done this to have a basic understanding of how things are going to work in this lab.

Line wrap

```
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>

```

5. Come back to Jupyter Notebook and start writing your code. First, start by importing requests, and we are defining the type(res) for the requests.

6. The code uses the **requests** library in Python to send an HTTP GET request to the website **"http://www.example.com"** and stores the response in the variable **res**. The type(res) command checks the type of the response object, which is typically requests.models.Response. Finally, res.text extracts and returns the content of the webpage as a string, usually containing the HTML structure of the page. If there are network restrictions or firewalls, the request may fail, requiring multiple attempts.
7. In cell 8, you can see that we got the source in the form of a string and we can change that using the beautiful soup from bs4 library.

```
[2]: import requests

[3]: # Step 1: Use the requests library to grab the page
     # Note, this may fail if you have a firewall blocking Python/Jupyter
     # Note sometimes you need to run this twice if it fails the first time
     res = requests.get("http://www.example.com")

[4]: type(res)

[4]: requests.models.Response

[8]: res.text

[8]: '<!doctype html>\n<html>\n<head>\n    <title>Example Domain</title>\n\n    <meta charset="utf-8" />\n    <meta http-equiv="Content-type" content="text/ht
     ml; charset=utf-8" />\n    <meta name="viewport" content="width=device-width, initial-scale=1" />\n    <style type="text/css">\n    body {\n        backg
     round-color: #f0f0f2;\n        margin: 0;\n        padding: 0;\n        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open San
     s", "Helvetica Neue", Helvetica, Arial, sans-serif;\n        \n    }\n    div {\n        width: 600px;\n        margin: 5em auto;\n        padding: 2e
     m;\n        background-color: #fdfdff;\n        border-radius: 0.5em;\n        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);\n    }\n    a:link, a:visite
     d {\n        color: #38488f;\n        text-decoration: none;\n    }\n    @media (max-width: 700px) {\n        div {\n            margin: 0 auto;\n
     width: auto;\n        }\n    }\n    </style>    \n</head>\n\n<body>\n<div>\n    <h1>Example Domain</h1>\n    <p>This domain is for use in illustrative ex
     amples in documents. You may use this\n    domain in literature without prior coordination or asking for permission.</p>\n    <p><a href="https://www.ian
     a.org/domains/example">More information...</a></p>\n</div>\n</body>\n</html>\n'
```

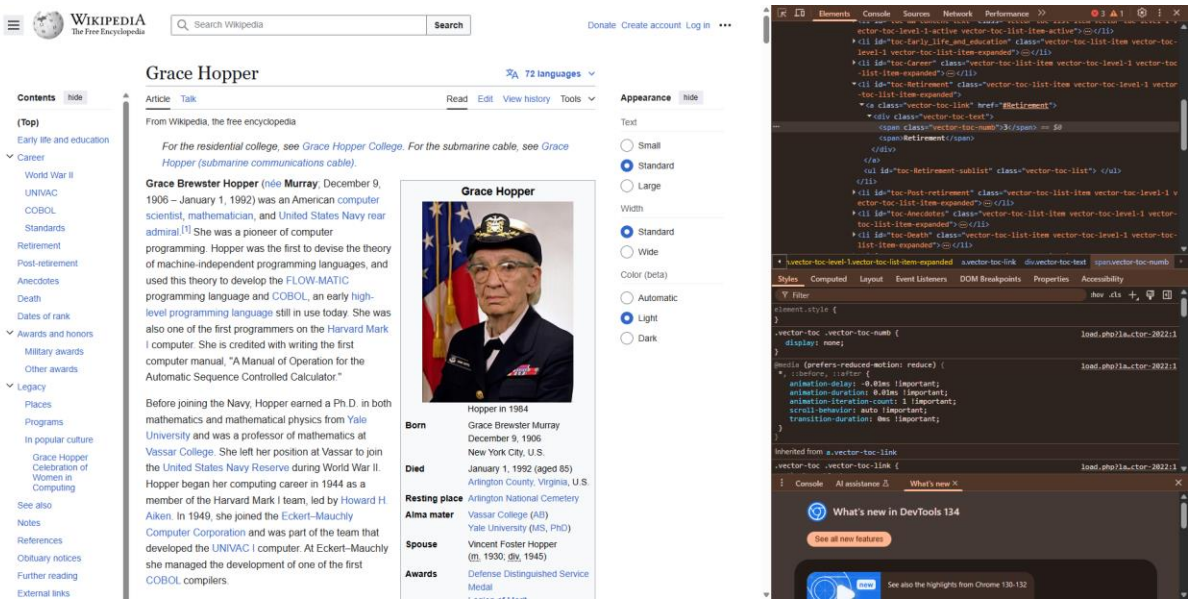
8. The code uses the **BeautifulSoup** library from **bs4** to parse the HTML content retrieved from the previous request (res.text). The **"lxml"** parser is specified for efficient and fast parsing. The resulting soup object represents the structured HTML document, allowing easy navigation, searching, and manipulation of elements within the webpage.

```
[10]: import bs4

[12]: soup = bs4.BeautifulSoup(res.text,"lxml")

[14]: soup

[14]: <!DOCTYPE html>
      <html>
      <head>
      <title>Example Domain</title>
      <meta charset="utf-8"/>
      <meta content="text/html; charset=utf-8" http-equiv="Content-type"/>
      <meta content="width=device-width, initial-scale=1" name="viewport"/>
      <style type="text/css">
          body {
              background-color: #f0f0f2;
              margin: 0;
              padding: 0;
              font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

          }
          div {
              width: 600px;
              margin: 5em auto;
```

9. The code extracts the <title> tag from the parsed HTML using soup.select('title'), which returns a list of matching elements. The first element (title_tag[0]) is accessed, confirming its type as a **Tag** object. Finally, title_tag[0].getText() retrieves the text content inside the <title> tag, effectively extracting the page's title.

```
[16]: soup.select('title')

[16]: [<title>Example Domain</title>]

[18]: title_tag = soup.select('title')

[20]: title_tag[0]

[20]: <title>Example Domain</title>

[22]: type(title_tag[0])

[22]: bs4.element.Tag

[24]: title_tag[0].getText()

[24]: 'Example Domain'
```

10. Now we are going to **Grab all the elements of a class,** for that we are going to use a Wikipedia page link https://en.wikipedia.org/wiki/Grace_Hopper.
11. If you go to this page and open the inspect menu by right-clicking in any white space area of the page then you will be able to see the elements of the page.



12. The code retrieves the HTML content of the Wikipedia page for **Grace Hopper** using the requests library and stores the response. It then parses the HTML using BeautifulSoup with the **lxml** parser, creating a soup object. This object represents the entire webpage structure, allowing easy extraction and manipulation of its elements, such as text, links, and tables.

```
[26]: # First get the request
      res = requests.get('https://en.wikipedia.org/wiki/Grace_Hopper')

[28]: # Create a soup from request
      soup = bs4.BeautifulSoup(res.text,"lxml")

[30]: soup
```

```
[30]: <!DOCTYPE html>
<html class="client-nojs vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-page-tools-pinned
-disabled vector-feature-toc-pinned-clientpref-1 vector-feature-main-menu-pinned-disabled vector-feature-limited-width-clientpref-1 vector-feature-limit
ed-width-content-enabled vector-feature-custom-font-size-clientpref-1 vector-feature-appearance-pinned-clientpref-1 vector-feature-night-mode-enabled sk
in-theme-clientpref-day vector-sticky-header-enabled vector-toc-available" dir="ltr" lang="en">
<head>
<meta charset="utf-8"/>
<title>Grace Hopper - Wikipedia</title>
<script>(function(){var className="client-js vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-featu
re-page-tools-pinned-disabled vector-feature-toc-pinned-clientpref-1 vector-feature-main-menu-pinned-disabled vector-feature-limited-width-clientpref-1
vector-feature-limited-width-content-enabled vector-feature-custom-font-size-clientpref-1 vector-feature-appearance-pinned-clientpref-1 vector-feature-n
ight-mode-enabled skin-theme-clientpref-day vector-sticky-header-enabled vector-toc-available";var cookie=document.cookie.match(/(?:^|; )enwikimwclientp
references=([^;]+)/);if(cookie){cookie[1].split('%2C').forEach(function(pref){className=className.replace(new RegExp('(^| )'+pref.replace(/-clientpref-
\w+$|[^\w-]+/g,'')+'-clientpref-\\w+( |$)'),'$1'+pref+'$2');});}document.documentElement.className=className;}());RLCONF={"wgBreakFrames":false,"wgSepar
atorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","January","February","March","April","May","J
une","July","August","September","October","November","December"],"wgRequestId":"97b025c1-91b9-4f1e-97a1-555401da1ab7","wgCanonicalNamespace":"","wgCano
nicalSpecialPageName":false,"wgNamespaceNumber":0,"wgPageName":"Grace_Hopper","wgTitle":"Grace Hopper","wgCurRevisionId":1283235419,"wgRevisionId":12832
35419,"wgArticleId":12590,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUserName":null,"wgUserGroups":["*"],"wgCategories":["All pages ne
```

13. The code attempts to extract all elements from the Wikipedia page that have the class **"vector-toc-numb"** using soup.select(). This class is commonly associated with **table of contents numbering** on Wikipedia. However, the exact class name might vary based on the Wikipedia page's structure or depending on regional variations of the site. If the class is different, the selector may return an empty list.

```
[34]: # note depending on your IP Address,
      # this class may be called something different
      soup.select(".vector-toc-numb")
```

```
[34]: [<span class="vector-toc-numb">1</span>,
       <span class="vector-toc-numb">2</span>,
       <span class="vector-toc-numb">2.1</span>,
       <span class="vector-toc-numb">2.2</span>,
       <span class="vector-toc-numb">2.3</span>,
       <span class="vector-toc-numb">2.4</span>,
       <span class="vector-toc-numb">3</span>,
       <span class="vector-toc-numb">4</span>,
       <span class="vector-toc-numb">5</span>,
       <span class="vector-toc-numb">6</span>,
       <span class="vector-toc-numb">7</span>,
       <span class="vector-toc-numb">8</span>,
       <span class="vector-toc-numb">8.1</span>,
       <span class="vector-toc-numb">8.2</span>,
       <span class="vector-toc-numb">9</span>,
       <span class="vector-toc-numb">9.1</span>,
       <span class="vector-toc-numb">9.2</span>,
       <span class="vector-toc-numb">9.3</span>
```

14. This code extracts and prints the text content of all HTML elements with the class "vector-toc-numb" from the Wikipedia page on Grace Hopper. These elements likely represent section numbers in the table of contents. The soup.select() method finds all matching elements, and the loop iterates over them, printing their text values.

```
[36]: for item in soup.select(".vector-toc-numb"):
          print(item.text)
```

```
1
2
2.1
2.2
2.3
2.4
3
4
5
6
7
8
8.1
8.2
9
9.1
9.2
9.3
```

15. Now we are going to get an image from a Wikipedia page, the initial steps are same as before.

```
[38]: res = requests.get("https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)")

[40]: soup = bs4.BeautifulSoup(res.text,'lxml')

[50]: soup
```

```
[50]: <!DOCTYPE html>
<html class="client-nojs vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-page-tools-pinned
-disabled vector-feature-toc-pinned-clientpref-1 vector-feature-main-menu-pinned-disabled vector-feature-limited-width-clientpref-1 vector-feature-limit
ed-width-content-enabled vector-feature-custom-font-size-clientpref-1 vector-feature-appearance-pinned-clientpref-1 vector-feature-night-mode-enabled sk
in-theme-clientpref-day vector-sticky-header-enabled vector-toc-available" dir="ltr" lang="en">
<head>
<meta charset="utf-8"/>
<title>Deep Blue (chess computer) - Wikipedia</title>
<script>(function(){var className="client-js vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-featu
re-page-tools-pinned-disabled vector-feature-toc-pinned-clientpref-1 vector-feature-main-menu-pinned-disabled vector-feature-limited-width-clientpref-1
vector-feature-limited-width-content-enabled vector-feature-custom-font-size-clientpref-1 vector-feature-appearance-pinned-clientpref-1 vector-feature-n
ight-mode-enabled skin-theme-clientpref-day vector-sticky-header-enabled vector-toc-available";var cookie=document.cookie.match(/(?:^|; )enwikimwclientp
references=([^;]+)/);if(cookie){cookie[1].split('%2C').forEach(function(pref){className=className.replace(new RegExp('(^| )'+pref.replace(/-clientpref-
\w+$|[^\w-]+/g,'')+'-clientpref-\\w+( |$)'),'$1'+pref+'$2');});}document.documentElement.className=className;}());RLCONF={"wgBreakFrames":false,"wgSepar
atorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","January","February","March","April","May","J
une","July","August","September","October","November","December"],"wgRequestId":"65f34b14-050d-4681-860f-539cdf461348","wgCanonicalNamespace":"","wgCano
nicalSpecialPageName":false,"wgNamespaceNumber":0,"wgPageName":"Deep_Blue_(chess_computer)","wgTitle":"Deep Blue (chess computer)","wgCurRevisionId":128
0401401,"wgRevisionId":1280401401,"wgArticleId":49387,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUserName":null,"wgUserGroups":
```

16. The code searches the parsed HTML (soup) for elements with the class .mw-file-element. In Wikipedia pages, this class is commonly used for images or media files embedded in the article. The output will be a list of elements (typically <img> tags) that match this class, allowing further extraction of image URLs or related attributes.

```
[60]: soup.select('.mw-file-element')
```

```
[60]: [<img alt="This is a good article. Click here for more information." class="mw-file-element" data-file-height="185" data-file-width="180" decoding="async" height="20" src="//upload.wikimedia.org/wikipedia/en/thumb/9/94/Symbol_support_vote.svg/19px-Symbol_support_vote.svg.png" srcset="//upload.wikimedia.org/wikipedia/en/thumb/9/94/Symbol_support_vote.svg/29px-Symbol_support_vote.svg.png 1.5x, //upload.wikimedia.org/wikipedia/en/thumb/9/94/Symbol_support_vote.svg/39px-Symbol_support_vote.svg.png 2x" width="19"/>,
 <img class="mw-file-element" data-file-height="601" data-file-width="400" decoding="async" height="331" src="//upload.wikimedia.org/wikipedia/commons/thumb/b/be/Deep_Blue.jpg/250px-Deep_Blue.jpg" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/b/be/Deep_Blue.jpg/330px-Deep_Blue.jpg 1.5x, //upload.wikimedia.org/wikipedia/commons/b/be/Deep_Blue.jpg 2x" width="220"/>,
 <img class="mw-file-element" data-file-height="64" data-file-width="64" decoding="async" height="150" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/52/Chess_Programming.svg/150px-Chess_Programming.svg.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/5/52/Chess_Programming.svg/225px-Chess_Programming.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/5/52/Chess_Programming.svg/300px-Chess_Programming.svg.png 2x" width="150"/>,
 <img class="mw-file-element" data-file-height="600" data-file-width="800" decoding="async" height="165" src="//upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Kasparov_Magath_1985_Hamburg-2.png/220px-Kasparov_Magath_1985_Hamburg-2.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Kasparov_Magath_1985_Hamburg-2.png/330px-Kasparov_Magath_1985_Hamburg-2.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Kasparov_Magath_1985_Hamburg-2.png/440px-Kasparov_Magath_1985_Hamburg-2.png 2x" width="220"/>,
 <img class="mw-file-element" data-file-height="2756" data-file-width="2067" decoding="async" height="293" src="//upload.wikimedia.org/wikipedia/commons/thumb/8/83/One_of_Deep_Blue%27s_processors_%282586060990%29.jpg/220px-One_of_Deep_Blue%27s_processors_%282586060990%29.jpg" srcset="//upload.wikimedia
```

17. This code extracts the second image element from the Wikipedia page using the **.mw-file-element class**, retrieves its **src** attribute (the image URL), and then downloads the image using requests.get(). The image_link.content holds the raw binary data of the image, which can be saved to a file using binary write mode (wb).

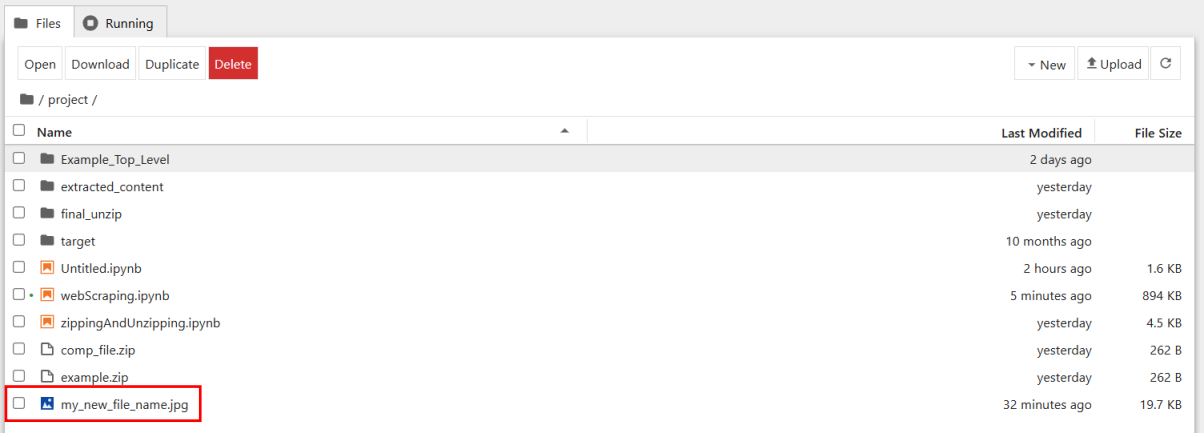
```
[86]: image_info = soup.select('.mw-file-element') [1]

[108]: image_info['src']

[108]: '//upload.wikimedia.org/wikipedia/commons/thumb/b/be/Deep_Blue.jpg/250px-Deep_Blue.jpg'

[114]: image_link = requests.get('https://upload.wikimedia.org/wikipedia/commons/thumb/b/be/Deep_Blue.jpg/250px-Deep_Blue.jpg')

[116]: # The raw content (its a binary file, meaning we will need to use binary read/write methods for saving it)
       image_link.content
```


```
[116]: b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x00H\x00H\x00\x00\xff\xfe\x00CFile source: http://commons.wikimedia.org/wiki/File:Deep_Blue.jpg\xff\xe2\x02@ICC_PROFILE\x00\x01\x01\x00\x00\x020ADBE\x02\x10\x00\x00mntrRGB XYZ \x07\xcf\x00\x06\x00\x03\x00\x00\x00\x00\x00\x00acspAPPL\x00\x00\x00\x00none\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xf6\xd6\x00\x01\x00\x00\x00\x00\xd3-ADBE\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00ncprt\x00\x00\x00\xfc\x00\x00\x002desc\x00\x00\x010\x00\x00\x00kwtpt\x00\x00\x01\x9c\x00\x00\x00\x14bkpt\x00\x00\x01\xb0\x00\x00\x00\x14rTRC\x00\x00\x01\xc4\x00\x00\x00\x0egTRC\x00\x00\x01\xd4\x00\x00\x00\x0ebTRC\x00\x00\x01\xe4\x00\x00\x00\x0erXYZ\x00\x00\x01\xf4\x00\x00\x00\x14gXYZ\x00\x00\x02\x08\x00\x00\x00\x14bXYZ\x00\x00\x02\x1c\x00\x00\x00\x14text\x00\x00\x00\x00Copyright 1999 Adobe Systems Incorporated\x00\x00\x00desc\x00\x00\x00\x00\x00\x00\x00\x11Adobe RGB (1998)\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00XYZ \x00\x00\x00\x00\x00\x00\xf3Q\x00\x01\x00\x00\x00\x01\x16\xccXYZ \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00curv\x00\x00\x00\x00\x00\x00\x00\x01\x023\x00\x00curv\x00\x00\x00\x00\x00\x00\x00\x01\x023\x00\x00curv\x00\x00\x00\x00\x00\x00\x00\x01\x023\x00\x00XYZ \x00\x00\x00\x00\x00\x00\x9c\x18\x00\x00O\xa5\x00\x00\x04\xfcXYZ \x00\x00\x00\x00\x00\x004\x8d\x00\x00\xa0,\x00\x00\x0f\x95XYZ \x00\x00\x00\x00\x00\x00&1\x00\x00\x10/\x00\x00\xbe\x9c\xff\xdb\x00C\x00\x06\x04\x05\x06\x05\x04\x06\x06\x05\x06\x07\x07\x06\x08\n\x10\n\n\t\t\n\x14\x0e\x0f\x0c\x10\x17\x14\x18\x18\x17\x14\x16\x16\x1a\x1d%\x1f\x1a\x1b#\x1c\x16\x16 , #&\')*)\x19\x1f-0-(0%()(\xff\xdb\x00C\x01\x07\x07\x07\n\x08\n\x13\n\n\x13(\x1a\x16\x1a((((((((((((((((((((((((((((((((((((((((((((((((((\xff\xc0\x00\x11\x08\x01x\x00\xfa\x03\x01\x11\x00\x02\x11\x01\x03\x11\x01\xff\xc4\x00\x1d\x00\x00\x00\x07\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x02\x03\x04\x05\x07\x08\x06\t\xff\xc4\x00X\x10\x00\x01\x03\x02\x03\x05\x04\x05\x06\x08\t\x08\x08\x07\x00\x00\x01\x02\x03\x11\x00\x04\x05!1\x06\x12AQa\x07\x13"q2\x81\x91\xa1\xb1\x08\x14#B\xc1\xd1\x15$34Rbr\xe1%Scs\x92\xa2\xa3\xb2\xf0\x
```


18. Now, using the code below, we open a new file named my_new_file_name.jpg in **binary write mode (wb)**, write the downloaded image data (image_link.content) into the file, and then close the file to ensure proper saving. This effectively stores the image on your local system.

```
[118]: f = open('my_new_file_name.jpg','wb')

[120]: f.write(image_link.content)

[120]: 20190

[122]: f.close()
```

19. So, in your current directory of Jupyter Notebook, you will see that an image has been created or downloaded. Now open this image.



20. Now we are moving to another example that is working with multiple pages and items.
21. The code below scrapes book listings from **Books to Scrape**, a test website for web scraping practice. It constructs a **URL template** (base_url) for different pages by using .format(). Then, it sends a **GET request** to fetch page 1, converts the response into a **BeautifulSoup object**, and selects all elements with the class .product_pod, which represent individual books.

```
[124]: base_url = 'http://books.toscrape.com/catalogue/page-{}.html'

[126]: res = requests.get(base_url.format('1'))

[128]: soup = bs4.BeautifulSoup(res.text,"lxml")

[130]: soup.select(".product_pod")
```

```
[130]: [<article class="product_pod">
        <div class="image_container">
        <a href="a-light-in-the-attic_1000/index.html"><img alt="A Light in the Attic" class="thumbnail" src="../media/cache/2c/da/2cdad67c44b002e7ead0cc35693c
       0e8b.jpg"/></a>
        </div>
        <p class="star-rating Three">
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        </p>
        <h3><a href="a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a></h3>
        <div class="product_price">
        <p class="price_color">Â£51.77</p>
        <p class="instock availability">
        <i class="icon-ok"></i>
```

22. This code extracts and analyzes book products from the **Books to Scrape** website.
23. It selects all elements with the class .product_pod, representing book listings. The first product is stored in example, and its attributes are examined using .attrs.
24. Finally, list(example.children) retrieves all direct child elements within the book's HTML structure, helping understand its internal layout for further data extraction.

```
[132]: products = soup.select(".product_pod")

[134]: example = products[0]

[136]: type(example)

[136]: bs4.element.Tag

[138]: example.attrs

[138]: {'class': ['product_pod']}

[140]: list(example.children)
```

```
[140]: ['\n',
        <div class="image_container">
        <a href="a-light-in-the-attic_1000/index.html"><img alt="A Light in the Attic" class="thumbnail" src="../media/cache/2c/da/2cdad67c44b002e7ead0cc35693c
       0e8b.jpg"/></a>
        </div>,
        '\n',
        <p class="star-rating Three">
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        </p>,
        '\n',
        <h3><a href="a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a></h3>,
        '\n',
        <div class="product_price">
        <p class="price_color">Â£51.77</p>
```

25. The code analyzes the **Books to Scrape** website's HTML structure to extract information about book ratings and titles. It searches for books with a **three-star or two-star rating** using CSS class selectors. Then, it finds all <a> (anchor) tags within a product listing, selects the second anchor tag, and extracts the **book title** from its title attribute, providing insight into the book's details.
26. In the end, we extracted the book titles that have a two-star rating from the book to scrape website.

```
[142]: example.select('.star-rating.Three')
```

```
[142]: [<p class="star-rating Three">
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        <i class="icon-star"></i>
        </p>]
```

```
[144]: example.select('.star-rating.Two')
```

```
[144]: []
```

```
[146]: example.select('a')
```

```
[146]: [<a href="a-light-in-the-attic_1000/index.html"><img alt="A Light in the Attic" class="thumbnail" src="../media/cache/2c/da/2cdad67c44b002e7ead0cc35693c0
       e8b.jpg"/></a>,
        <a href="a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a>]
```

```
[148]: example.select('a')[1]
```

```
[148]: <a href="a-light-in-the-attic_1000/index.html" title="A Light in the Attic">A Light in the ...</a>
```

```
[150]: example.select('a')[1]['title']
```

```
[150]: 'A Light in the Attic'
```

```
[154]: two_star_titles
```

```
[154]: ['Starving Hearts (Triangular Trade Trilogy, #1)',
        'Libertarianism for Beginners',
        "It's Only the Himalayas",
        'How Music Works',
        'Maude (1883-1993):She Grew Up with the country',
        "You can't bury them all: Poems",
        'Reasons to Stay Alive',
        'Without Borders (Wanderlove #1)',
        'Soul Reader',
        'Security',
        'Saga, Volume 5 (Saga (Collected Editions) #5)',
        'Reskilling America: Learning to Labor in the Twenty-First Century',
        'Political Suicide: Missteps, Peccadilloes, Bad Calls, Backroom Hijinx, Sordid Pasts, Rotten Breaks, and Just Plain Dumb Mistakes in the Annals of Amer
       ican Politics',
        'Obsidian (Lux #1)',
        'My Paris Kitchen: Recipes and Stories',
        'Masks and Shadows',
        'Lumberjanes, Vol. 2: Friendship to the Max (Lumberjanes #5-8)'
```