Web Scraping

HTML for Data Science

HTML (HyperText Markup Language) is the standard language to create web pages and structure content on the web.

These web pages are structured using **tags**, which define elements like headings, paragraphs, links, tables, lists, and forms. When our web page loads, our browser interprets HTML to display this content accordingly.

HTML Syntax

```
<tagname>content</tagname>
```

HTML documents consist of **elements** defined by **tags.**

Basic HTML Structure

```
<!DOCTYPE html>
<html>
<head>
    <title>My First Page</title>
</head>
<body>
    <h1>Welcome to PRIME!</h1>
    <p>This is a paragraph of text.</p>
</body>
</html>
```

Explanation:

- `<!DOCTYPE html>` - Defines the document as HTML5.
- `<html>` - Root element of the page.
- `<head>` - Contains metadata, page title, scripts, styles etc.
- `<body>` - Visible content of the page.
- `<h1>` - Heading (largest).
- `<p>` - Paragraph.

Important HTML Tags for Web Scraping

When scraping, these are the main elements we'll encounter:

Tag	Meaning	Use
`<h1>` to `<h6>`	Heading	Headings (important for titles, article sections)
`<p>`	Paragraph	Paragraphs (article or description text)
`<a>`	Anchor (link)	Hyperlinks (can extract URLs)
`<img>`	Image	Images (scrape image URLs from `src` attribute)
`<table>`	Table	Tabular data (rows = `<tr>`, columns = `<td>` / `<th>`)
`<ul>` / `<ol>` / `<li>`	List	Lists of items
`<div>` / `<span>`	Container	Generic containers, often used with CSS classes for scraping
`<form>` / `<button>` / `<input>`	Form related	Form related elements (for Flask Input)

Example:

```
<div>
    <h2>This is a demo content about HTML.</h2>
    <ol>
        <li>HTML is NOT case-sensitive.</li>
        <li>It is NOT a programming language but a markup language.</li>
    </ol>

    <p>Search on Google:
        <a href="www.google.com"> </a>
    </p>
</div>
```

Note – HTML follows a hierarchical structure, so tags can contain other tags.

Attributes in HTML

HTML elements can have **attributes** that store additional information like metadata or instructions.

Some common attributes are:

- `href` - link URL in `<a>` tag
- `src` - source of image in `<img>` tag
- `id` - unique identifier
- `class` - group of elements (used heavily in scraping)
- `type` - form element to define type of input
- `name` - assigns a name to a form element
- `value` - represents data associated with form element

Example:

```
<a href="https://example.com" id="link1" class="external">
    Visit Example
</a>
```

Example of a **Table** in HTML:

```
<table>
    <tr>
        <th>Name</th>
        <th>Marks</th>
    </tr>
    <tr>
        <td>Adam</td>
        <td>95</td>
    </tr>
    <tr>
        <td>Bob</td>
        <td>88</td>
    </tr>
</table>
```

Example of a **Form** in HTML:

```
<form action="/submit" method="POST">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">
    <input type="submit" value="Submit">
</form>
```

Web Scraping

There are many popular Python libraries for web scraping, the most commonly used ones are:

1. `requests` - Used to download the webpage i.e. HTML
2. `BeautifulSoup` - Parses & extracts data from HTML (useful for static HTML)
   We use `requests` & `BeautifulSoup` together.
3. `Selenium` - Automate a browser to load dynamic JS pages - optional alternative to (requests + BeautifulSoup) in case of dynamic websites.

Working with `requests`

The `requests` module in Python is a library that lets us send HTTP requests to websites or APIs. We can get, post, download, and send data over the internet easily.

Link to Documentation: https://pypi.org/project/requests/

Install (if needed):

```
pip install requests
```

Import

```
import requests
```

Basic GET Request

GET request is used to retrieve a webpage or API response.

```
import requests

URL = "https://example.com"
res = requests.get(URL)

print(res.status_code) # HTTP status (200 = OK)
print(res.text)        # HTML content of the page as a string
print(res.content)     # Raw bytes of the response
print(res.headers)     # Metadata like content type, server info etc.
```

Common Status Codes

Code	Meaning
200	OK
301/302	Redirect
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Server Error

ALWAYS check for `status_code` before working with data:

```
URL = "https://example.com"
res = requests.get(URL)

if res.status_code == 200:
  print(res.text)
else:
  print("Failed to Fetch page")
```

Downloading & Storing to Files

```
response = requests.get("https://example.com")

with open("scraped_data/data.html", "w") as f:
    f.write(res.text)
```

Note -

1. Just like GET requests, we also have POST, DELETE & other requests which are out of scope for our course but GET is going to be the most relevant for us for now (in context of web scraping).
2. We can also use `try-except` blocks to handle exception(s) in our code, like this:

```
try:
    r = requests.get("https://example.com", timeout=5)
    r.raise_for_status()    # Raises error for 4xx/5xx responses
except requests.exceptions.RequestException as e:
    print("Error:", e)
```

Headers (Useful in Web Scraping)

Websites often expect certain headers, like “User-Agent” that lets them know that the request is coming from real browser. This can help us avoid getting blocked when sending automated requests.

```
headers = {
    "User-Agent": "Mozilla/5.0"
}

response = requests.get("https://example.com", headers=headers)
print(response.text)
```

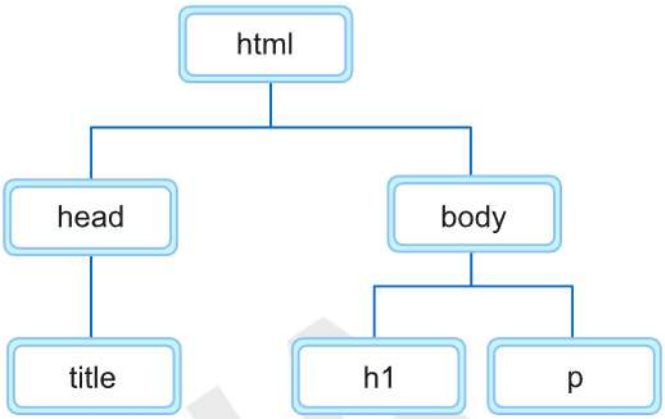
We can also introduce delays using `time.sleep()` in scraping to avoid hitting servers too fast.

Working with `BeautifulSoup`

BeautifulSoup (bs4) is a Python library used to parse HTML and XML documents. It helps you extract data from webpages after downloading them with `requests`.

It creates a **Parse tree** from the data from which we can extract text, attributes, links, tables, etc.

Example of a Parse Tree Structure



Install

```
pip install beautifulsoup4
```

Import

```
from bs4 import BeautifulSoup
```

Keep Learning & Keep Exploring!