Day-11

Introduction to Web Scraping

Web scraping is the automated process of extracting information from websites. This information can be stored in various formats such as databases, spreadsheets, or plain text files. Web scraping allows for the collection of large amounts of data from the internet efficiently and quickly.

Web Crawling vs. Web Scraping

- **Web Crawling**: This involves systematically browsing the web and downloading web pages. A web crawler, also known as a spider, navigates from page to page via links, indexing content for search engines.
- **Web Scraping**: This focuses on extracting specific data from web pages. Scrapers usually access web pages directly and extract relevant information, bypassing the need to index the entire site.

Uses of Web Scraping

- **Market Research**: Collecting data from competitor websites.
- **Content Aggregation**: Gathering news articles, blog posts, or other content types.
- **Price Monitoring**: Tracking product prices from various e-commerce sites.
- **Data Analysis**: Collecting data for machine learning and data science projects.
- **Research**: Gathering data for academic or scientific research.

Components of a Web Scraper

- **HTTP Requests**: Used to request web pages from servers.
- **HTML Parsing**: Extracting relevant data from the HTML of web pages.
- **Data Storage**: Storing the extracted data in a structured format (e.g., CSV, database).
- **Scheduler**: Automating the scraping process to run at specific intervals.

Working of a Web Scraper

1. **Send Request:** The scraper sends an HTTP request to the target website.
2. **Download HTML:** The server responds with the HTML content of the webpage.
3. **Parse HTML:** The scraper parses the HTML to extract the desired data.
4. **Transform Data:** The extracted data is cleaned and transformed as needed.
5. **Store Data:** The cleaned data is stored in a structured format for further use.

Crawl, Parse, Transform, and Store the Data

1. **Crawl**: Use libraries like `requests` to send HTTP requests and retrieve web pages.
2. **Parse**: Use libraries like Beautiful Soup to parse the HTML content.
3. **Transform**: Clean and manipulate the data as required.
4. **Store**: Save the data to a CSV file, database, or other storage formats.

Beautiful Soup

Introduction to Beautiful Soup Library

Beautiful Soup is a Python library for parsing HTML and XML documents. It creates a parse tree for parsing HTML and XML documents, making it easy to extract data from them.

Installing Beautiful Soup

You can install Beautiful Soup using pip:

```
!pip install beautifulsoup4
```

Import:

```
# Step 1: Import required libraries
import requests
from bs4 import BeautifulSoup
```

Request and response:

```
import requests
from bs4 import BeautifulSoup

response = requests.get("https://www.niet.co.in/")
print(response.content)
```

Using beautifulSoup, html.parser and prettify()

```python
# Step 1: Import required libraries
import requests
from bs4 import BeautifulSoup

# Step 2: Send an HTTP request to get the HTML content of the web page
url = 'https://www.niet.co.in/'
response = requests.get(url)
html_content = response.content

# Step 3: Parse HTML content
soup = BeautifulSoup(html_content, 'html.parser')
print(soup.prettify())
```

Example 1: Extracting Table Data from < title > Tags

```python
import requests
from bs4 import BeautifulSoup
response = requests.get('https://www.niet.co.in/')
soup = BeautifulSoup(response.content,'html.parser')
print(soup.title)
print(soup.title.string)
```

```
<title>Top Engineering Colleges in Greater Noida | Best Placement Institute in Delhi NCR UP</title>
Top Engineering Colleges in Greater Noida | Best Placement Institute in Delhi NCR UP
```

 # Example 2: Extracting Text from < p > Tags

```
from bs4 import BeautifulSoup

# HTML content
html_content = '''
<html>
<head><title>Example Page</title></head>
<body>
<p class="paragraph">This is the first paragraph.</p>
<p class="paragraph">This is the second paragraph.</p>
</body>
</html>
'''

# Parse the HTML content
soup = BeautifulSoup(html_content, 'html.parser')

# Find all <p> tags
paragraphs = soup.find_all('p')
print(paragraphs)         # SEE OUTPUT
# Extract and print text from each <p> tag
for p in paragraphs:
    print(p.get_text())
```

```
[<p class="paragraph">This is the first paragraph.</p>, <p class="paragraph">This is the second paragraph.</p>]
This is the first paragraph.
This is the second paragraph.
```

Example 3: Extracting Attributes from < a > Tags

```
from bs4 import BeautifulSoup

# HTML content
html_content = '''
<html>
<head><title>Example Page</title></head>
<body>
<a href="https://example.com/page1" class="link">Page 1</a>
<a href="https://example.com/page2" class="link">Page 2</a>
</body>
</html>
'''

# Parse the HTML content
soup = BeautifulSoup(html_content, 'html.parser')

# Find all <a> tags
links = soup.find_all('a')

# Extract and print href attribute from each <a> tag
for link in links:
    print(link['href'])
```

```
https://example.com/page1
https://example.com/page2
```

Example 4: Extracting Text and Attributes from and <img> Tags

```
from bs4 import BeautifulSoup

# HTML content
html_content = '''
<html>
<head><title>Example Page</title></head>
<body>
<div class="profile">
    <h2>John Doe</h2>
    <img src="profile.jpg" alt="John's Profile Picture">
</div>
</body>
</html>
'''

# Parse the HTML content
soup = BeautifulSoup(html_content, 'html.parser')

# Find the <div> with class 'profile'
profile_div = soup.find('div', class_='profile')

# Extract and print the text from <h2> tag inside the profile <div>
name = profile_div.find('h2').get_text()
print(f'Name: {name}')

# Extract and print the src attribute from <img> tag inside the profile <div>
profile_image = profile_div.find('img')['src']
print(f'Profile Image: {profile_image}')
```

```
Name: John Doe
Profile Image: profile.jpg
```

Example 5: Extracting Text and Attributes from < ul > Tags

```
from bs4 import BeautifulSoup

# HTML content
html_content = '''
<html>
<head><title>Example Page</title></head>
<body>
<ul class="items">
    <li>Python</li>
    <li>Advanced Python</li>
    <li>C Programming</li>
</ul>
</body>
</html>
'''

# Parse the HTML content
soup = BeautifulSoup(html_content, 'html.parser')

# Find the <ul> with class 'items'
items_list = soup.find('ul', class_='items')

# Extract and print text from each <li> tag inside the <ul>
items = items_list.find_all('li')
for item in items:
    print(item.get_text())
```

```
Python
Advanced Python
C Programming
```

Example 6: Extracting Table Data from <table> Tags

```
from bs4 import BeautifulSoup

# HTML content
html_content = '''
<html>
<head><title>Example Page</title></head>
<body>
<table class="data">
    <tr><th>Header 1</th><th>Header 2</th></tr>
    <tr><td>Row 1, Cell 1</td><td>Row 1, Cell 2</td></tr>
    <tr><td>Row 2, Cell 1</td><td>Row 2, Cell 2</td></tr>
</table>
</body>
</html>
'''

# Parse the HTML content
soup = BeautifulSoup(html_content, 'html.parser')

# Find the <table> with class 'data'
table = soup.find('table', class_='data')

# Extract and print header cells
headers = table.find_all('th')
header_texts = [header.get_text() for header in headers]
print(f'Headers: {header_texts}')
```

```
# Extract and print row data
rows = table.find_all('tr')[1:]  # Skip the header row
for row in rows:
    cells = row.find_all('td')
    cell_texts = [cell.get_text() for cell in cells]
    print(f'Row: {cell_texts}')
```

```
Headers: ['Header 1', 'Header 2']
Row: ['Row 1, Cell 1', 'Row 1, Cell 2']
Row: ['Row 2, Cell 1', 'Row 2, Cell 2']
```

Introduction to GitHub

What is GitHub?

GitHub is a web-based platform used for version control and collaboration. It allows multiple people to work on projects simultaneously while tracking changes and maintaining a history of revisions.

Key Features

- **Repositories**: Storage spaces for project files and their history.
- **Branches**: Separate lines of development within a repository.
- **Commits**: Snapshots of changes made to files in a repository.
- **Pull Requests**: Proposed changes to a repository, allowing for review and discussion before merging.
- **Issues**: Tracking bugs, enhancements, and tasks.

Basic Workflow

1. **Create a Repository**: Initialize a new repository for your project.
2. **Clone the Repository**: Copy the repository to your local machine.

    git clone https://github.com/username/repository.git

3. **Make Changes:** Modify files and stage your changes.

```
git add file1 file2
git commit -m "Description of changes"
```

4. **Push Changes:** Upload your changes to the remote repository.

```
git push origin main
```