Web Scraping with Beautiful Soup and Requests

Web scraping is the process of extracting data from websites. Python provides powerful libraries like **Beautiful Soup** (for parsing HTML and XML) and **requests** (for making HTTP requests) to facilitate web scraping.

---

1. Install Required Libraries

Before starting, ensure that you have the required libraries installed:

bash
Copy code
```
pip install beautifulsoup4 requests
```

---

2. Importing Libraries

python
Copy code
```
import requests
from bs4 import BeautifulSoup
```

---

3. Fetching a Webpage

Use the `requests` library to send an HTTP GET request to a webpage:

python
Copy code
```
url = 'https://example.com'  # Replace with the target URL
response = requests.get(url)

if response.status_code == 200:
    print("Successfully fetched the webpage!")
else:
    print(f"Failed to fetch the webpage. Status code:
{response.status_code}")

# Print the first 500 characters of the HTML content
print(response.text[:500])
```

---

4. Parsing HTML with Beautiful Soup

python
Copy code

```
# Create a BeautifulSoup object
soup = BeautifulSoup(response.text, 'html.parser')

# Print the formatted HTML (prettified version)
print(soup.prettify()[:500])
```

---

5. Extracting Data

Find Elements by Tag

python
Copy code

```
# Extract the title of the page
title = soup.title.string
print(f"Page Title: {title}")

# Find all paragraph tags
paragraphs = soup.find_all('p')
for p in paragraphs[:5]:  # Limit to first 5 paragraphs
    print(p.text)
```

Find by ID or Class

python
Copy code

```
# Find an element by its ID
div_with_id = soup.find('div', id='example-id')
print(div_with_id.text if div_with_id else "ID not found")

# Find elements by class name
divs_with_class = soup.find_all('div', class_='example-class')
for div in divs_with_class[:5]:
    print(div.text)
```

Extract Links

python
Copy code

```
# Extract all hyperlinks
links = soup.find_all('a')
```

```
for link in links[:10]:  # Limit to first 10 links
    print(link.get('href'))
```

Extract Images

python
Copy code

```
# Extract all image URLs
images = soup.find_all('img')
for img in images[:5]:  # Limit to first 5 images
    print(img.get('src'))
```

---

6. Navigating the HTML Tree

Parent, Children, and Siblings

python
Copy code

```
# Access parent of an element
first_paragraph = soup.find('p')
print(first_paragraph.parent.name)

# Access children of a tag
body = soup.body
for child in body.children:
    print(child.name)

# Access next and previous siblings
next_element = first_paragraph.next_sibling
print(next_element)
```

---

7. Filtering with CSS Selectors

Beautiful Soup supports CSS selectors for precise searches.

python
Copy code

```
# Select elements using CSS selectors
elements = soup.select('div.example-class p')
for element in elements:
    print(element.text)
```

---

8. Handling Pagination

When scraping multiple pages:

python
Copy code

```
for page in range(1, 6):  # Scrape first 5 pages
    url = f'https://example.com/page={page}'
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extract desired data from the current page
    items = soup.find_all('div', class_='item')
    for item in items:
        print(item.text)
```

---

9. Saving Extracted Data

You can save the scraped data into a CSV or JSON file:

python
Copy code

```
import csv

data = [{'Title': 'Example 1', 'URL': 'https://example1.com'},
        {'Title': 'Example 2', 'URL': 'https://example2.com'}]

# Save to CSV
with open('scraped_data.csv', mode='w', newline='',
encoding='utf-8') as file:
    writer = csv.DictWriter(file, fieldnames=['Title', 'URL'])
    writer.writeheader()
    writer.writerows(data)
```

---

10. Ethical Considerations

- **Check Website Terms of Service**: Ensure the website allows scraping.

Use a User-Agent: Mimic a browser request.
python
Copy code
```
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36'}
response = requests.get(url, headers=headers)
```

-

Respect Rate Limits: Avoid sending too many requests in a short time.
python
Copy code
```
import time
time.sleep(1)  # Delay between requests
```

-

---

11. Advanced Techniques

- **Working with APIs**: If the website provides an API, use it instead of scraping.
- **Dynamic Websites**: Use Selenium or Playwright for JavaScript-rendered content.

Error Handling:
python
Copy code
```
try:
    response = requests.get(url)
    response.raise_for_status()
except requests.exceptions.RequestException as e:
    print(f"Error: {e}")
```

-