Web Scraping in Python

Pankaj Chouhan
www.codeswithpankaj.com

What is Web Scraping ?

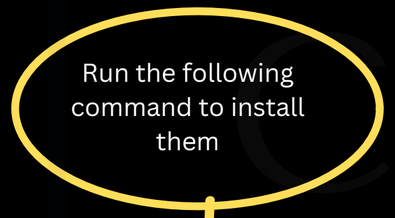
Web scraping is a technique to extract information from websites. It is used to collect data from web pages automatically.

Libraries for Web Scraping

- **Requests :** Fetches web pages.
- **BeautifulSoup :** Extracts data from HTML.

Prerequisites

Install Required Libraries




```
!pip install requests beautifulsoup4
%pip install requests beautifulsoup4
```

Import Required Libraries

```
import requests
from bs4 import BeautifulSoup
```

Step by step. We will

- ✅ Fetch a webpage using requests
- ✅ Parse and extract data using BeautifulSoup
- ✅ Extract specific information like blog titles and links
- ✅ Save data to a CSV file

Step 1: Install Required Libraries

```
pip install requests beautifulsoup4 lxml pandas
```

- **requests →** To fetch the website HTML
- **beautifulsoup4 →** To parse and extract data from the HTML
- **lxml →** To improve parsing performance
- **pandas →** To save the data in a structured format

Step 2: Fetch the Website HTML

Let's send a request to www.codeswithpankaj.com and retrieve the page content.

```
import requests

url = "https://www.codeswithpankaj.com/"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}

response = requests.get(url, headers=headers)

# Check if the request was successful
if response.status_code == 200:
    print("Page fetched successfully!")
    print(response.text[:500])  # Print first 500 characters of the HTML
else:
    print(f"Failed to fetch page. Status code: {response.status_code}")
```

Explanation :

- ✓ We use requests.get(url) to fetch the webpage.
- ✓ We add a User-Agent header to prevent blocking.
- ✓ If the request is successful (status_code == 200), we print a portion of the HTML.

Step 3: Parse the HTML with BeautifulSoup

Once we get the page content, we parse it using BeautifulSoup.

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(response.text, "html.parser")

# Print the page title
print("Page Title:", soup.title.string)
```

Explanation :

- ✓ We pass the HTML content to BeautifulSoup for parsing.
- ✓ soup.title.string extracts the webpage title.

Pankaj Chouhan
www.codeswithpankaj.com

Step 4: Extract Blog Titles & Links

Now, let’s extract the latest blog post titles and their links from the homepage.

```
# Find all blog post titles and links
articles = soup.find_all("h2", class_="post-title")

for article in articles:
    title = article.text.strip()
    link = article.a["href"]
    print(f"Title: {title}")
    print(f"Link: {link}\n")
```

Explanation :

- ✓ soup.find_all("h2", class_="post-title") finds all <h2> elements with the class post-title.
- ✓ We extract the blog title and link using .text.strip() and .a["href"].

Step 5: Save Data to CSV

Let’s store the extracted data in a CSV file.

```
import csv

# Open CSV file to save data
with open("blog_posts.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["Title", "Link"])  # Column headers

    # Loop through extracted articles
    for article in articles:
        title = article.text.strip()
        link = article.a["href"]
        writer.writerow([title, link])

print("Data saved to blog_posts.csv")
```

Explanation :

- ✓ We open a CSV file in write mode.
- ✓ We write column headers: "Title", "Link".
- ✓ We loop through the extracted data and store it in the file.

Step 6: Handling Pagination (Multiple Pages)

If the website has multiple pages, we can scrape all pages by looping through them.

```
page = 1
all_posts = []

while True:
    url = f"https://www.codeswithpankaj.com/page/{page}/"
    response = requests.get(url, headers=headers)

    if response.status_code != 200:
        break  # Stop if no more pages

    soup = BeautifulSoup(response.text, "html.parser")
    articles = soup.find_all("h2", class_="post-title")

    if not articles:
        break  # Stop if no more blog posts

    for article in articles:
        title = article.text.strip()
        link = article.a["href"]
        all_posts.append([title, link])

    page += 1

# Save to CSV
with open("all_blog_posts.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow(["Title", "Link"])
    writer.writerows(all_posts)

print(f"Scraped {len(all_posts)} blog posts and saved to all_blog_posts.csv")
```

Explanation :

- ✓ We loop through pages (/page/1/, /page/2/, etc.).
- ✓ If no articles are found, we stop scraping.
- ✓ We save all posts to all_blog_posts.csv.

Step 7: Avoid Getting Blocked

Websites may block scrapers if they detect too many requests. Here's how to avoid that:

```
import time
import random

time.sleep(random.randint(2, 5))
# Wait 2-5 seconds before the next request
```

Use Random Delays

```
import random

user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)",
    "Mozilla/5.0 (X11; Ubuntu; Linux x86_64)"
]

headers = {"User-Agent": random.choice(user_agents)}
```

Rotate User-Agents

Pankaj Chouhan
www.codeswithpankaj.com

Final Thoughts

🎉 Congratulations! You've successfully scraped www.codeswithpankaj.com.

- ✅ Fetched the webpage
- ✅ Extracted blog titles & links
- ✅ Saved data to a CSV file
- ✅ Scraped multiple pages
- ✅ Avoided getting blocked

More projects

Pankaj Chouhan
www.codeswithpankaj.com