National University of Computer and Emerging Sciences

DL-2001: Introduction to Data Science
Lab Manual 05

Department of Data Science
FAST-NU, Lahore, Pakistan

1. Objective

To understand and implement web scraping using **Python's BeautifulSoup library** to extract data from a website.

2. Prerequisites

- Basic knowledge of Python programming
- Understanding of HTML structure
- Familiarity with installing Python packages

3. Background Theory

3.1 What is Web Scraping?

- Web scraping is the automation of extracting data from websites. It involves retrieving HTML from a web server, and parsing it to pull out information.
- Manual copying or browsing is web scraping, but usually "web scraping" refers to doing this programmatically.

3.2 Challenges of Web Scraping

- **Variety**: HTML structures vary between websites; you often must adapt your parsing to each site.
- **Durability**: Sites change; CSS classes, HTML layout, or naming might shift, breaking scrapers.
- **Dynamic Content & JavaScript**: Some sites don't send all content in the static HTML; some content is generated client‑side via JS. Plain requests + BeautifulSoup won't fetch dynamic content.
- **Login / Authentication**: Some data is behind login pages. Handling cookies, sessions, or other authentication is more complex.
- **Legal/Ethical Considerations**: Respect robots.txt, terms of service; don't overload servers; ensure you are allowed to use the data.

3.3 APIs as an Alternative

- APIs often provide structured data (JSON, XML) directly. More stable, often more legal / permitted, less brittle when site front-ends change.

- However, APIs may have rate limits, require authentication, or not offer all publicly visible data.

3.4 Preparing / Inspecting the Data Source

1. **Choose a site** you can legally scrape; static content is easier.
2. **Explore the site** with browser tools: click through pages, see how content is structured.
3. **Understand URLs**: base URL, path, query parameters; knowing how URLs change when you navigate/search helps.
4. **Use Developer Tools** (Inspect / Elements panel) to see HTML structure: IDs, class names, element nesting.

3. 5 What is BeautifulSoup?

BeautifulSoup is a powerful Python library used to **parse HTML and XML documents**. It provides simple methods to **navigate, search, and modify the parse tree** (i.e., the structure of an HTML page), making it easier to extract information from web pages.
It is most commonly used in **web scraping**, where data is extracted from websites by parsing their HTML content.

Why Use BeautifulSoup?

- Parses HTML and XML **quickly and cleanly**
- Works with **complex, nested HTML structures**
- Offers an easy-to-use API to search elements by **tags, classes, attributes, text**, etc.
- Can be combined with requests to fetch live web data

Feature	Description
Tag searching	Find elements using tag names (e.g., , <a>)
Attribute filtering	Find tags with specific id, class, or other attributes
Text extraction	Get the text content of any HTML element
Tree navigation	Move between parent, children, siblings in the HTML DOM
Multiple parsers	Supports different parsers like html.parser, lxml, etc.

How to Install

```
pip install beautifulsoup4
```

Basic Example:

```
from bs4 import BeautifulSoup

html_code = """
<html>
  <body>
    <h1>My Website</h1>
    <p class='intro'>Welcome to my website</p>
  </body>
</html>
"""

# Create a BeautifulSoup object
soup = BeautifulSoup(html_code, 'html.parser')

# Extract the title
print(soup.h1.text)  # Output: My Website

# Find the paragraph with class 'intro'
intro = soup.find('p', class_='intro')
print(intro.text)  # Output: Welcome to my website
```

Commonly Used methods:

Method	Description
soup.find()	Returns the **first matching tag**
soup.find_all()	Returns **all matching tags** in a list
tag.text	Extracts the **text content** of a tag
tag['href'] or .get('href')	Extracts **attribute values**, like URLs
soup.select()	Finds elements using **CSS selectors** (advanced)

Additional Resources

- Official docs: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

4. Step-By-Step: Building the Scraper using Beautifulsoup

Here is a guided set of steps, drawn from the Real Python tutorial, to build a web scraper.

Step 1: Fetch HTML Content

- Use requests.get(URL) to fetch page.
- Use .content rather than .text when passing to BeautifulSoup, to handle encoding more robustly.

```
import requests

from bs4 import BeautifulSoup

URL = "https://realpython.github.io/fake-jobs/"

page = requests.get(URL)
```

Step 2: Parse HTML with BeautifulSoup

- Create a BeautifulSoup object with the fetched HTML:

```
soup = BeautifulSoup(page.content, "html.parser")
```

- This object allows you to navigate the HTML document, search for elements, etc.

Step 3: Find Specific Elements

3.1 Finding by ID

- If there is a container element with a known ID (e.g. "ResultsContainer"), you can do:

```
results = soup.find(id="ResultsContainer")
```

This locates that part of the HTML to narrow down search.

3.2 Finding by Class Name

- Within results, find all the job posting cards by class:

```
job_cards = results.find_all("div", class_="card-content")
```

- Each job_card now corresponds to one job posting.

3.3 Extracting Text

- From each job_card, extract the needed fields (title, company, location):

```
for card in job_cards:
    title = card.find("h2", class_="title").text.strip()
    company = card.find("h3", class_="company").text.strip()
    location = card.find("p", class_="location").text.strip()
    print(title, company, location)
```

- Use .strip() to clean up whitespace.

Step 4: Filtering Based on Text

- If you want only jobs with “Python” in the title, you can do:

```
python_jobs = results.find_all(
    "h2",
    string=lambda text: "python" in text.lower()
)
```

- This picks <h2> tags where the title string contains “python” (case-insensitive).
- But remember: such filtered tags may not include the rest of the job info, so you may need to move up to parent elements.

Step 5: Navigating Parent / Child Relationships

- Once you identify a tag (e.g. <h2>), you can get its parent, grandparent, etc., to find the container that holds all related info:

```
python_job_cards = [
    h2_element.parent.parent.parent
    for h2_element in python_jobs
]
```

- Then from each such job card, you can find the other elements (company, location, etc.).

Step 6: Extracting Attributes (URLs etc.)

- Sometimes you want not just visible text, but attributes like href from <a> tags. For example, to fetch the "Apply" link:

```
link_url = job_card.find_all("a")[1]["href"]
print(f"Apply here: {link_url}")
```

- Use indexing when multiple links in each card; use square-bracket notation to access attributes.

Step 7: Putting Everything Together (Script)

Here's a full script putting the pieces together.

```
import requests

from bs4 import BeautifulSoup



URL = "https://realpython.github.io/fake-jobs/"

page = requests.get(URL)

soup = BeautifulSoup(page.content, "html.parser")



results = soup.find(id="ResultsContainer")



python_jobs = results.find_all(

    "h2",

    string=lambda text: "python" in text.lower()

)



python_job_cards = [

    h2_element.parent.parent.parent

    for h2_element in python_jobs

]
```

```
for job_card in python_job_cards:
    title_element = job_card.find("h2", class_="title")
    company_element = job_card.find("h3", class_="company")
    location_element = job_card.find("p", class_="location")
    print(title_element.text.strip())
    print(company_element.text.strip())
    print(location_element.text.strip())
    link_url = job_card.find_all("a")[1]["href"]
    print(f"Apply here: {link_url}\n")
```

Reference link: https://realpython.com/beautiful-soup-web-scraper-python/