Web Scraping and HTML Basics

Estimated time: 10 mins

Lab Objectives:

By the end of this reading, you should be able to:

1. Understand key concepts related to HTML structure.
2. Learn about HTML tag composition.
3. Explore the concept of HTML document trees.
4. Familiarize yourself with HTML tables.
5. Gain insight into the basics of web scraping using Python and BeautifulSoup.

Introduction to Web scrapping

Web scraping, also known as web harvesting or web data extraction, is the process of extracting information from websites or web pages. It involves automated retrieval of data from web sources and is commonly used for a wide range of applications such as data analysis, data mining, price comparison, content aggregation, and more.

How web scraping works:

HTTP Request:

The process typically begins with an HTTP request. A web scraper sends an HTTP request to a specific URL, similar to how a web browser would when you visit a website. The request is usually an HTTP GET request, which retrieves the content of the web page.

Web Page Retrieval:

The web server hosting the website responds to the request by sending back the requested web page's HTML content. This content includes not only the visible text and media elements but also the underlying HTML structure that defines the page's layout.

HTML Parsing:

Once the HTML content is received, it needs to be parsed. Parsing involves breaking down the HTML structure into its individual components, such as tags, attributes, and text content. This is where a library like BeautifulSoup in Python is commonly used. It creates a structured representation of the HTML content that can be easily navigated and manipulated.

Data Extraction:

With the HTML content parsed, web scrapers can now identify and extract the specific data they need. This data can include text, links, images, tables, product prices, news articles, and more. Scrapers locate the data by searching for relevant HTML tags, attributes, and patterns in the HTML structure.

Data Transformation:

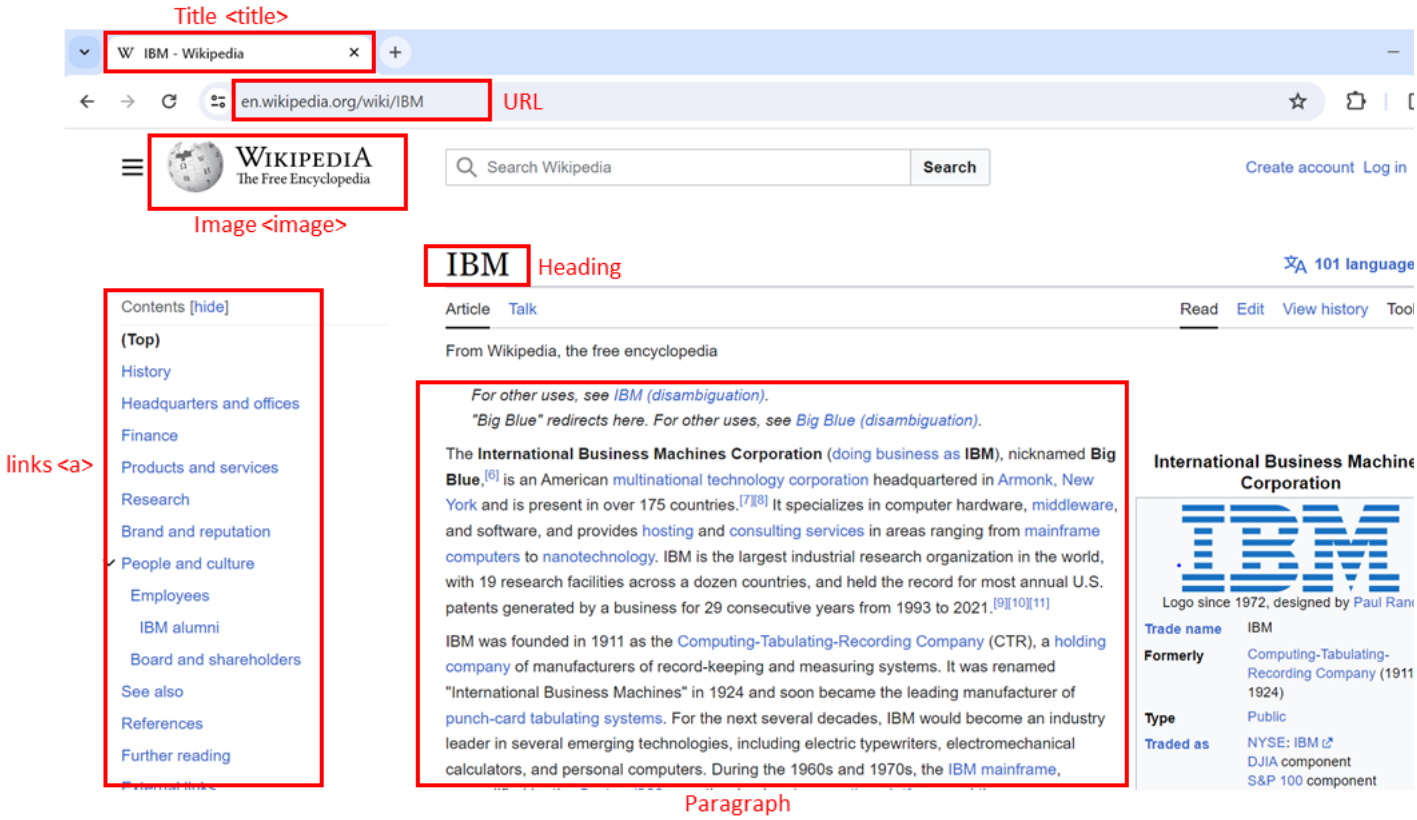
Extracted data may need further processing and transformation. For instance, removing HTML tags from text, converting data formats, or cleaning up messy data. This step ensures that the data is ready for analysis or other use cases.

Storage:

After extraction and transformation, the scraped data can be stored in various formats, such as databases, spreadsheets, or even JSON or CSV files. The choice of storage format depends on the specific project's requirements.

Automation:

In many cases, web scraping is automated using scripts or programs. These automation tools allow for recurring data extraction from multiple web pages or websites. Automated scraping is especially useful for collecting data from dynamic websites that regularly update their content.



HTML Structure

HTML (Hypertext Markup Language) serves as the foundation of web pages. Understanding its structure is crucial for web scraping.

- `<html>` is the root element of an HTML page.
- `<head>` contains meta-information about the HTML page.
- `<body>` displays the content on the web page, often the data of interest.
- `<h3>` tags are type 3 headings, making text larger and bold, typically used for player names.
- `<p>` tags represent paragraphs and contain player salary information.

Composition of an HTML Tag

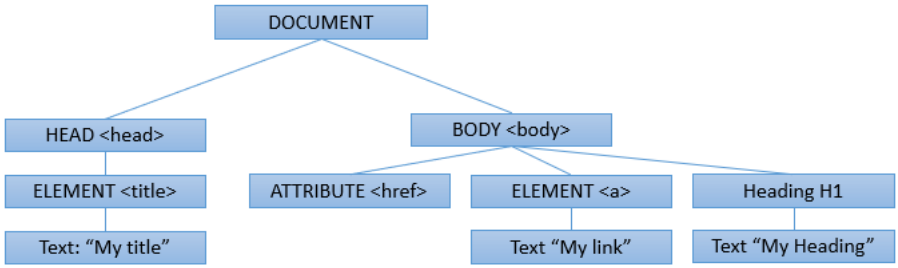
HTML tags define the structure of web content and can contain attributes.

- An HTML tag consists of an opening (start) tag and a closing (end) tag.
- Tags have names (e.g., `<a>` for an anchor tag).
- Tags may contain attributes with an attribute name and value, providing additional information to the tag.

HTML Document Tree

HTML documents can be visualized as trees with tags as nodes.

- Tags can contain strings and other tags, making them the tag's children.
- Tags within the same parent tag are considered siblings.
- For example, the `<html>` tag contains both `<head>` and `<body>` tags, making them descendants of `<html` but children of `<html>`. `<head>` and `<body>` are siblings.

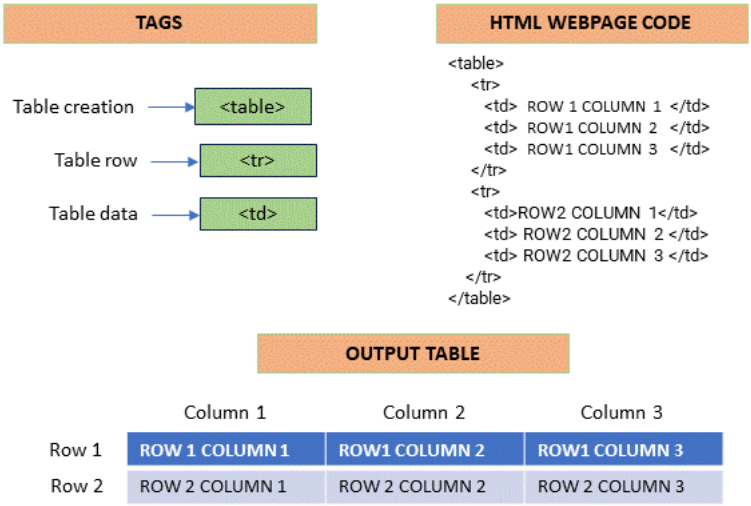


HTML Tables

HTML tables are essential for presenting structured data.

- Define an HTML table using the `<table>` tag.
- Each table row is defined with a `<tr>` tag.

- The first row often uses the table header tag, typically `<th>`.
- The table cell is represented by `<td>` tags, defining individual cells in a row.



Web Scraping

Web scraping involves extracting information from web pages using Python. It can save time and automate data collection.

Required Tools:

Web scraping requires Python code and two essential modules: Requests and Beautiful Soup. Ensure you have both modules installed in your Python environment.

```
1
2
```

```
# Import Beautiful Soup to parse web page content
from bs4 import BeautifulSoup
```

Copied!

Fetching and Parsing HTML:

To start web scraping, you need to fetch the HTML content of a webpage and parse it using Beautiful Soup. Here's a step-by-step example:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
import requests
from bs4 import BeautifulSoup

# Specify the URL of the webpage you want to scrape
url = 'https://en.wikipedia.org/wiki/IBM'

# Send an HTTP GET request to the webpage
response = requests.get(url)

# Store the HTML content in a variable
html_content = response.text

# Create a BeautifulSoup object to parse the HTML
soup = BeautifulSoup(html_content, 'html.parser')

# Display a snippet of the HTML content
print(html_content[:500])
```

Copied!

Navigating the HTML Structure:

BeautifulSoup represents HTML content as a tree-like structure, allowing for easy navigation. You can use methods like find_all to filter and extract specific HTML elements. For example, to find all anchor tags () and print their text:

```
1
2
3
4
```

```
5. 5
6. 6
```

```
1. # Find all <a> tags (anchor tags) in the HTML
2. links = soup.find_all('a')
3.
4. # Iterate through the list of links and print their text
5. for link in links:
6.     print(link.text)
```

Copied!

Custom Data Extraction:

Web scraping allows you to navigate the HTML structure and extract specific information based on your requirements. This may involve finding specific tags, attributes, or text content within the HTML document.

Using BeautifulSoup for HTML Parsing

Beautiful Soup is a powerful tool for navigating and extracting specific parts of a web page. It allows you to find elements based on their tags, attributes, or text, making it easier to extract the information you're interested in.

Using pandas read_html for Table Extraction

On many websites, data is neatly organized in tables. Pandas, a Python library, provides a function called read_html, which can automatically extract data from these tables and present it in a format suitable for analysis. It's similar to taking a table from a webpage and importing it into a spreadsheet for further analysis.

Conclusion

In summary, this reading introduces web scraping with BeautifulSoup and Pandas, emphasizing extracting elements and tables. BeautifulSoup facilitates HTML parsing, while Pandas' read_html streamlines table extraction. Responsible web scraping is highlighted, ensuring adherence to website terms. Armed with this knowledge, you can confidently engage in precise data extraction.

Author

Akansha Yadav

Changelog

Date	Version	Changed by	Change Description
2023-04-11	0.1	Akansha Yadav	Initial version created