Lecture 4

Data Collection II: Web-scrapping Primer; Scrapping Data with `selenium`

Byeong-Hak Choe

bchoe@geneseo.edu

SUNY Geneseo

February 13, 2026

Premier on Web-scrapping

🎣📥 Data Collection via Web-scraping

- Web pages can be a rich data source, but **web scraping is powerful**.
  - Careless scraping can **harm websites, violate rules, or compromise privacy**.
- Our goal in this module:
  - Learn the **web fundamentals** (client/server, HTTPS, URL, HTML/DOM),
  - Understand **ethical, responsible scraping**

⚖️🤔 “Legal” Is Not the Same as “Ethical”

> *“If you can see things in your web browser, you can scrape them.”*

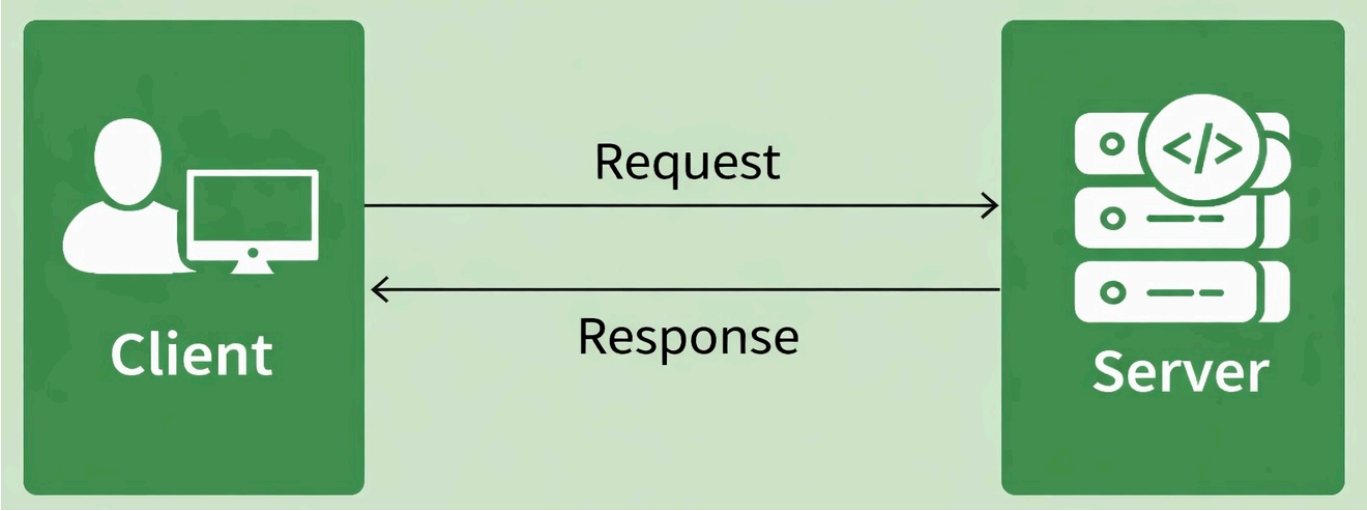
- *Legally (U.S.)*: **publicly available** data may sometimes be scraped using automated tools in US (e.g., **hiQ Labs vs. LinkedIn Corp.**)
- *But legality ≠ permission or responsibility*:
  - *Technically*: it may be possible.
  - *Ethically*: you still must consider terms or service (ToS), robots.txt, privacy, and data minimization.
  - *Practically*: you can trigger blocks or harm service quality (e.g., overloading servers, ToS/privacy issues).

Warning

Legal ≠ ethical. Even if data is “public,” ToS, privacy expectations, and platform blocks still matter.

Web Basics: Clients and Servers

💻↔🗄️ Clients and Servers



- Devices on the web act as **clients** and **servers**.
- Your browser is a **client**; websites and data live on **servers**.
  - **Client**: your computer/phone + a browser (Chrome/Firefox/Safari).
  - **Server**: a computer that stores webpages/data and sends them when requested.
- When you load a page, your browser sends a **request** and the server sends back a **response** (the page content).

🔒🛡️ Hypertext Transfer Protocol Secure (HTTPS)

- **HTTP** is how clients and servers communicate.
- **HTTPS** is encrypted HTTP (safer).

When we type a URL starting with `https://`:

1. Browser finds the server.
2. Browser and server establish a secure connection.
3. Browser sends a request for content.
4. Server responds (e.g., **200 OK**) and sends data.
5. Browser decrypts and displays the page.

🚦🔢 HTTP Status Codes

```
# library for making HTTPS requests in Python
import requests
```

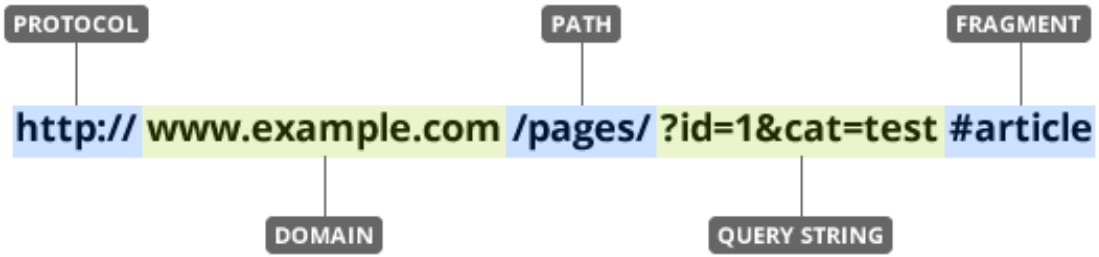
```
p = 'https://bcdanl.github.io/210'
response = requests.get(p)
print(response.status_code)
print(response.reason)
```

- **200 OK** → success; content returned.

```
p = 'https://bcdanl.github.io/2100'
response = requests.get(p)
print(response.status_code)
print(response.reason)
```

- **404 Not Found** → URL/page doesn't exist (typo, removed page, broken link).

🔗📍 URL (what you're actually requesting)



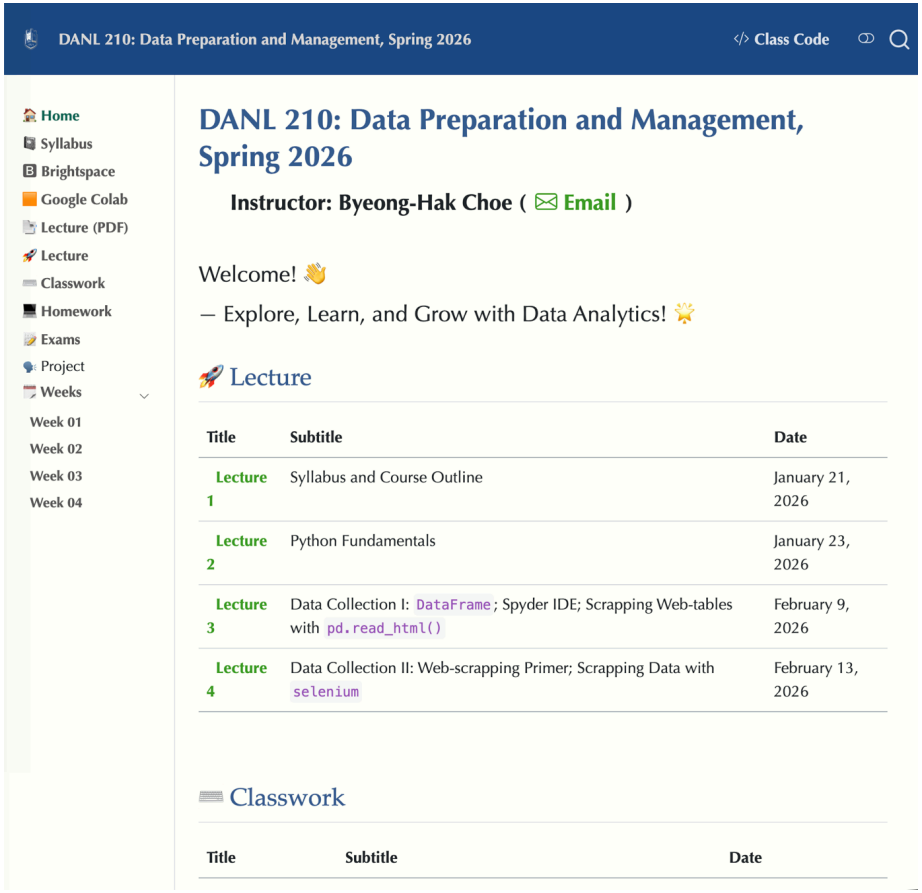
- A **URL** is a location for a resource on the internet.
- Often includes:
  - Protocol (`https`)
  - Domain (`example.com`)
  - Path (`/products`)
  - **Query string** (`?id=...&cat=...`) ← common in data pages
  - Fragment (`#section`) ← in-page reference

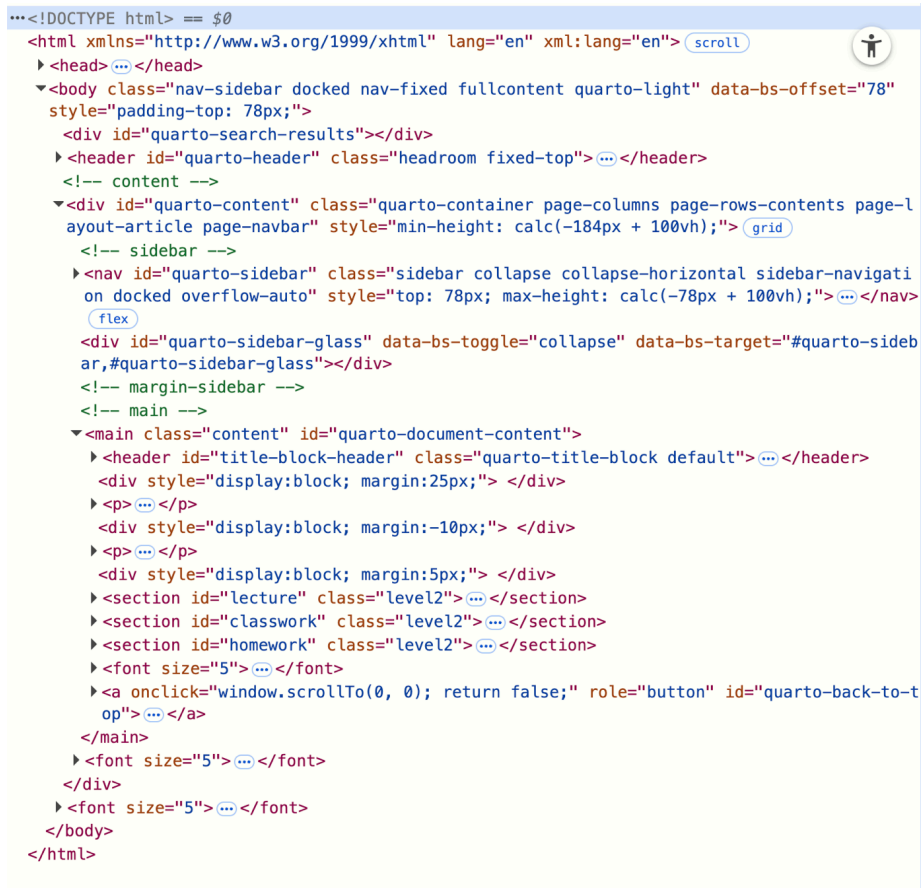
HTML Basics

🎯👈 The Big Idea: Scraping = Selecting from HTML

- **HTML** (HyperText Markup Language) is the markup that defines the **structure** of a web page (headings, paragraphs, links, tables, etc.).
- When you "scrape," you usually:
  1. Load a page
  2. Examine the **HTML**
  3. Extract specific elements (title, price, table, links, etc.)
- **If you don't understand HTML, you can't reliably target the right data.**
- Selenium is not "magic"—it automates a browser, but you still need to:
  - Inspect the HTML to identify and target the right elements

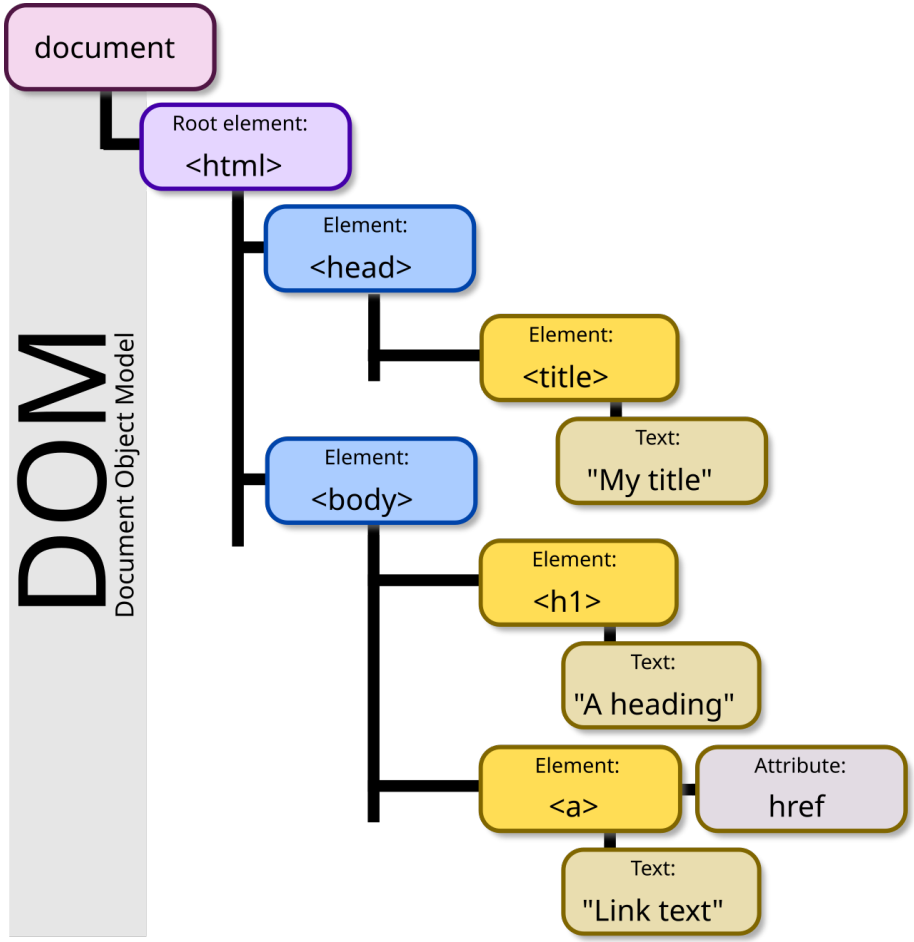
HTML in Browser vs. HTML Source Code





Document Object Model (DOM)

The Browser's "Tree" of the Page



- The browser represents HTML as the **DOM** (Document Object Model).
- Selenium interacts with the **DOM**.
- Scraping often becomes:
  - "Find the node"
  - "Extract its text/attribute"

🔍🕵️ Inspecting HTML (your #1 web-scrapping skill)

- Open a **Chrome** browser.
- Open DevTools:
  - **F12**, or right-click → **Inspect**
- Use it to find:
  - Element text
  - `id` / `class`
  - Attributes (like `href`, `data-*`)

🧱🧩 HTML Elements (what you actually scrape)

- Most HTML is built from **elements** like:

```
<tagname>Content goes here...</tagname>
```

- Common ones you'll extract:
  - Headings: `<h1> ... </h1>`
  - Text blocks: `<p> ... </p>`
  - Links: `<a href="..."> ... </a>`
  - Tables: `<table> ... </table>`
  - Containers: `<div> ... </div>`
  - Inline text: `<span> ... </span>`

🔗🖼️ HTML Body: Links and Images

<a> (Link)

```
1  <a href="https://www.w3schools.com">This is a link</a>
```

- The `href` attribute is often what you scrape.

<img> (Image)

```
1  <img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

- You may scrape `src` (image URL) or `alt` (description).

HTML Tables

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

- Table structure:
  - `<table>` table container
  - `<tr>` row
  - `<th>` header cell
  - `<td>` data cell

📋 Lists you'll see in the wild

⚫ Unordered List (<ul>)

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

- Coffee
- Tea
- Milk

🔢 Ordered List (<ol>)

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

1. Coffee
2. Tea
3. Milk

🎯📦 Containers you'll target a lot: `<div>` and `<span>`

`<div>` – block-level container

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>Seoul</h2>
  <p>Seoul is the capital city of South Korea...</p>
</div>
```

- Often used to group major page sections.

`<span>` – inline container

```
<p>My mother has <span style="color:blue;font-weight:bold">blue</span> eyes...</p>
```

My mother has **blue** eyes…