**Week 1: Introduction to Web Scraping and Basics**

**Day 1: Introduction to the Web and HTML**

* **What is a webpage?**
  + A webpage is like a digital page on the internet. Every page we see on a website is written in HTML (HyperText Markup Language).
* **What is HTML?**
  + HTML is the language used to structure the contents of a webpage. It uses **tags** to define different parts of the content. For example, <p> is used for paragraphs, and <h1> is used for headings.
* **What is the DOM (Document Object Model)?**
  + The DOM is a tree-like structure that shows how HTML elements are arranged on a webpage. Each tag is like a branch on this tree, and it helps us understand how the page is built.

**Day 2: Fetching Web Pages using Python (requests)**

* **What is HTTP?**
  + HTTP (HyperText Transfer Protocol) is how computers communicate with websites. It is like a messenger delivering a request to the website and bringing the webpage back to you.
* **What is the requests library?**
  + In Python, we use requests to fetch or "download" the HTML of a webpage. It is like sending a message to a website, asking for its page.
* **Response Codes:**
  + When we request a webpage, we get a **status code**:
    - **200**: Success, the webpage is available.
    - **404**: Not found, the webpage does not exist.

**Day 3: Parsing HTML using BeautifulSoup**

* **What is BeautifulSoup?**
  + BeautifulSoup is a Python library that helps us "read" and "understand" the HTML code of a webpage. It allows us to easily find and extract specific information from the page.
* **What does parsing mean?**
  + Parsing means taking the messy HTML code and breaking it into parts we can use. For example, we can extract only the text from inside specific tags, like paragraphs (<p>).

**Day 4: Extracting Specific Data from Web Pages**

* **How to find elements (5 main ways)**

**1. By tag name**

**```python**

**soup.find('a') first <a>**

**soup.find\_all('a') list of all <a>**

**```**

**Good for: simple pages, broad grabs.**

**2. By attributes (id, class, others)**

**```python**

**soup.find('div', id='main')**

**soup.find\_all('span', class\_='price') note: class\_ (underscore)**

**soup.find\_all('img', attrs={'alt': True, 'loading': 'lazy'})**

**```**

**Good for: stable IDs/classes; any HTML attribute via `attrs`.**

**3. CSS selectors (most flexible)**

**```python**

**soup.select('ulmenu > li.active a') all matching nodes**

**soup.select\_one('table.data tbody tr:nth-of-type(1) td:nth-of-type(2)')**

**```**

**Good for: complex structures, nth-child, combinators, attribute selectors.**

**4. Text-based search (exact or regex)**

**```python**

**from re import compile**

**To get the element containing that text:**

**el = soup.find(string='Contact Us').parent**

**soup.find\_all(string=compile(r'\bSale\b', flags=0))**

**```**

**Good for: headings/labels without clean classes.**

**5. Custom filter functions**

**```python**

**def is\_product\_link(tag):**

**return tag.name == 'a' and tag.get('href', '').startswith('/product/')**

**links = soup.find\_all(is\_product\_link)**

**```**

**Good for: any logic that’s hard to express with attributes/CSS.**

**Day 5: Practical Example - Scraping a Simple Website**

* **How do we use requests and BeautifulSoup together?**
  + First, we fetch the webpage using requests. Then, we use BeautifulSoup to parse the HTML and find the data we are interested in, like the title or paragraphs.

**Week 2: Advanced Web Scraping Techniques**

**Day 6: Scraping Multiple Pages**

* **What is pagination?**
  + Pagination means dividing a large number of items (like search results) into multiple pages. For example, search results on Google often show "Next" and "Previous" buttons to move between pages.
* **How do we scrape multiple pages?**
  + We use loops in Python to go through different pages, like "Page 1," "Page 2," etc. We change the URL in each loop and fetch the HTML for each page.

**Day 7: Handling Dynamic Content (Intro to JavaScript-based Pages)**

* **What is static vs. dynamic content?**
  + **Static content** is when everything is loaded when the page first opens. We can scrape it easily.
  + **Dynamic content** is when parts of the page are loaded later using JavaScript. These are harder to scrape because Python doesn’t run JavaScript on its own.
* **Why do we use Selenium for dynamic pages?**
  + Selenium is a tool that controls a web browser. It can open pages, click on buttons, and fill forms—this is useful when we need to interact with pages that load content dynamically using JavaScript.

**Day 8: Handling Headers and User Agents**

* **What are headers and user-agents?**
  + **Headers** are like the ID cards of a request. They tell the website who is asking for the page.
  + A **user-agent** tells the website what type of device or browser is making the request. For example, it might say you are using Chrome on a Windows computer.
* **Why change user-agents?**
  + Sometimes websites block scraping tools by recognizing their user-agent. By pretending to be a browser, we can avoid being blocked.

**Day 9: Introduction to Data Storage (Saving Scraped Data)**

* **Why save scraped data?**
  + After collecting data from websites, we often need to store it in a file so we can use it later. For example, we might store it in a **CSV file**, which is like a spreadsheet where we can keep our data organized.
* **What is a CSV file?**
  + **CSV (Comma-Separated Values)** is a simple format to store data in rows and columns, where each value is separated by a comma.

**Day 10: Final Project - Scraping and Data Analysis**

* **What is the final project?**
  + In this project, students will choose a website (like a news site or Wikipedia) and write a Python script to:
    - **Fetch and scrape specific data** (like article titles).
    - **Store the data in a file** (like a CSV).
    - **Present their results** (showing the first few entries in their data).