Artificial Intelligence Diploma

Python Session 9

NeuroTech

Agenda

1. What is Web Scraping?
2. HTML Basics
3. Extract Data from Web Site.
4. Practice.

What is Web Scraping

Web Scraping

- Web scraping is the process of extracting data from websites. It involves automatically fetching web pages, parsing their content, and extracting relevant information for various purposes. This technique enables you to gather data from websites in a structured format, which can be further analyzed, processed, and utilized for different applications.

Web Scraping

Web scraping is commonly used to:

1. **Data Collection:** Collect data like product prices, reviews, stock market data, weather information, news headlines, and more.
2. **Market Research:** Gather data about competitors, customer sentiment, and market trends for business insights.
3. **Content Aggregation:** Aggregate and display content from multiple sources on a single platform.
4. **Monitoring:** Monitor changes on websites for real-time updates, such as stock prices or social media mentions.
5. **Data Analysis:** Analyze web data for research, reporting, and decision-making.
6. **Lead Generation:** Extract contact information from websites for marketing purposes.
7. **Academic Research:** Collect data for research projects, surveys, and analysis.

Web Scraping

Web scraping involves several steps to extract data from websites effectively. Here are the detailed steps involved in web scraping:

1. **Define the Objective:**
   - Determine the purpose of your web scraping project. What data do you need to extract, and why?
2. **Choose a Programming Language:**
   - Python is a popular choice due to its powerful libraries for web scraping, such as requests, **BeautifulSoup**, **lxml**, and **Selenium**.

3. **Install Necessary Libraries:**
   - Install the required libraries using a package manager, such as pip for Python.
   - Example: pip install requests BeautifulSoup4 selenium (for Python)

Web Scraping

4. **Identify the Target Website:**
    - Choose the website(s) from which you want to scrape data.
5. **Analyze Website Structure:**
    - Inspect the target website's structure, including HTML and CSS.
    - Determine the location of the data you want to extract and the HTML tags and attributes associated with it.
6. **Send HTTP Requests:**
    - Use the requests library (or an equivalent in your chosen language) to send HTTP GET requests to the website's URLs.
    - Example: response = requests.get("https://example.com")
7. **Receive HTTP Responses:**
    - Capture the server's response, which typically contains HTML content.
8. **Parse HTML Content:**
    - Use an HTML parsing library like BeautifulSoup or lxml to parse and structure the HTML content.
    - Example: soup = BeautifulSoup(response.text, 'html.parser')

Web Scraping

9. Locate and Select Elements:

- •Identify the HTML elements (e.g., div, p, a) that contain the data you want to scrape.
- •Use selectors like CSS selectors or XPath to locate elements.
- •Example: titles = soup.select('.title')

10. Extract Data:

-Extract the desired data from the located elements. - Access text, attributes, or other content within the elements.

- Example: title_text = title.text

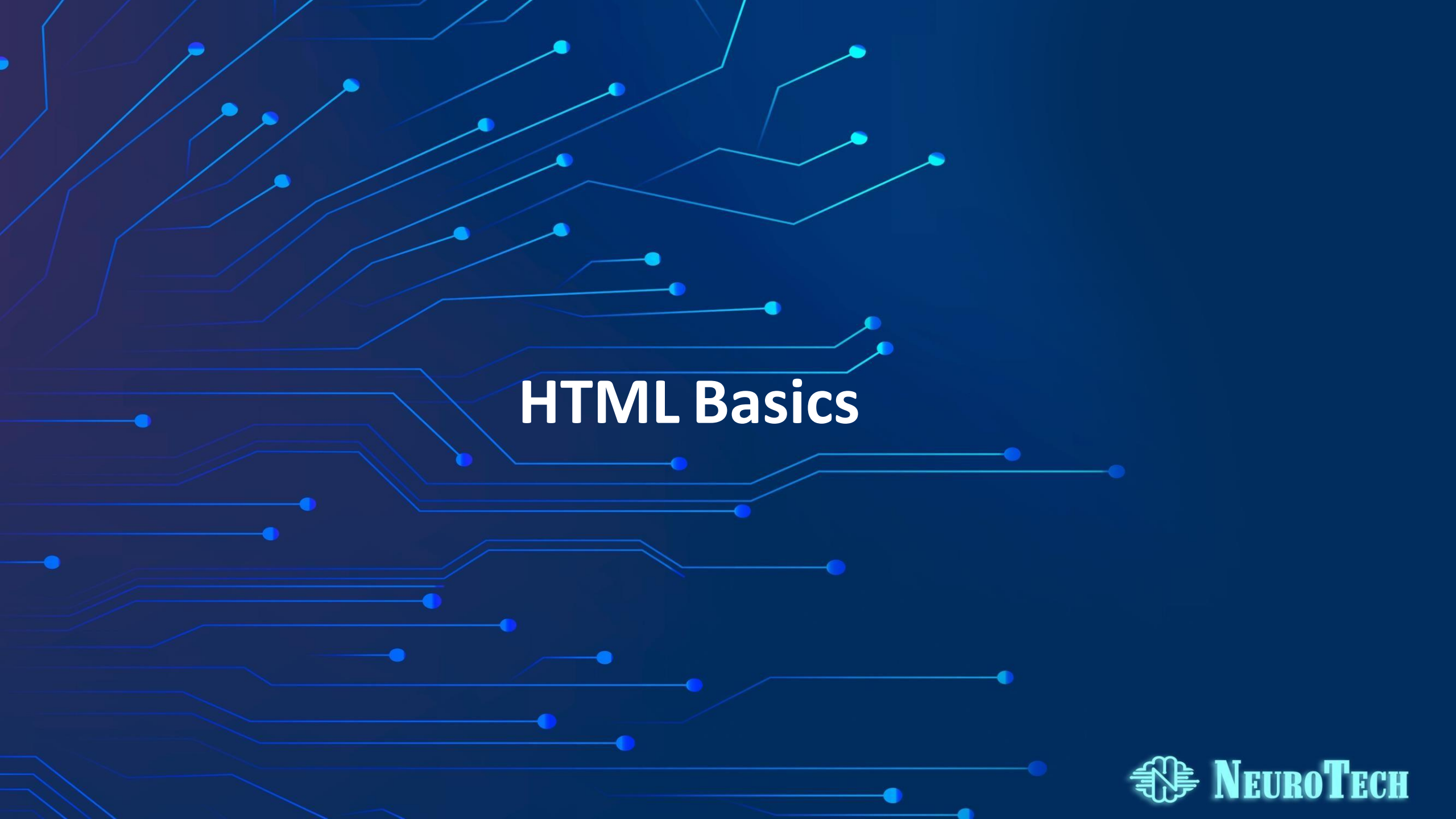
11. Clean and Structure Data:

- Clean and format the extracted data to make it usable and consistent.
- Remove unwanted characters, whitespace, or HTML tags.
- Structure the data into a suitable format, such as lists, dictionaries, or dataframes.

13. Store Data:

- Save the scraped data into a structured format, such as CSV, JSON, or a database.

HTML Basics



Structure of HTML documents

- HTML (Hypertext Markup Language) is the standard language used to create web pages. When it comes to web scraping, understanding HTML basics is essential, as it allows you to extract information from web pages by navigating through their structure.

- HTML document follows a specific structure that defines how content should be organized and displayed in a web browser. This structure consists of several essential components. Here's the basic structure of an HTML document:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Document Title</title>
</head>
<body>
    <!-- Content of the web page goes here -->
</body>
</html>
```

Structure of HTML documents

Let's break down each component:

- **DOCTYPE Declaration (<!DOCTYPE html>):** The DOCTYPE declaration defines the version of HTML being used. In modern web development, the HTML5 doctype is most commonly used. It informs the browser to interpret the document as HTML5.
- **<html> Element:** The <html> element is the root element of an HTML document. All other elements are contained within it.
- **<head> Element:** The <head> element contains metadata about the document, such as the character encoding, title, and links to external resources like stylesheets and scripts. It's not displayed on the actual web page.
  1. <meta charset="UTF-8">: Specifies the character encoding for the document. UTF-8 is commonly used to support a wide range of characters.
  2. <title>: Sets the title of the web page, which appears in the browser's title bar or tab.

Structure of HTML documents

- **<body> Element:** The <body> element contains the visible content of the web page, such as text, images, links, and other elements. This is where users interact with the content.

```
<body>
    <header>
        <h1>Welcome to My Website</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <article>
            <h2>Article Title</h2>
            <p>This is the content of the article...</p>
        </article>
    </main>

    <footer>
        <p>&copy; 2023 My Website. All rights reserved.</p>
    </footer>
</body>
```

HTML Basics

- **HTML Structure:** HTML documents are structured using a set of elements that define the content and layout of a web page. Elements are enclosed in opening and closing tags, and they can contain text, other elements, or both.

  Example:

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

- **Tags:** Tags are the building blocks of HTML. They indicate the beginning and end of elements and define the type of content within them.

  Example:

```
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
```

HTML Basics

- **Attributes:** HTML tags can have attributes that provide additional information about the element. Attributes are placed within the opening tag and consist of a name and a value.

  Example:

```
<a href="https://www.example.com">Visit Example</a>
```

- **Elements and Nesting:** HTML elements can be nested inside each other to create a hierarchical structure. The nesting order determines the relationship between elements.

  Example:

```
<div>
    <h2>Header</h2>
    <p>Some text</p>
</div>
```

HTML Basics

- **IDs and Classes:** IDs and classes are attributes that are often used to uniquely identify or group elements. IDs should be unique on a page, while classes can be applied to multiple elements.

  Example:

```
<div id="unique-id">This is a unique element.</div>
<p class="info">This is an informational paragraph.</p>
```

- **Comments:** HTML comments are used to provide explanations or notes within the code. They are not displayed on the web page.

  Example:

```
<!-- This is a comment -->
```

HTML Basics

Common HTML Elements:

- <h1> to <h6>: Headings
- <p>: Paragraph
- <a>: Anchor link
- <img>: Image
- : Unordered list
- : Ordered list
- : List item
- <table>: Table
- <tr>: Table row
- <td>: Table cell
- : Division or container
- : Inline container

Extract Data from Web Site

NeuroTech

Web Scraping

- Know We Provide step-by-step guide on how to extract data from a website using the **BeautifulSoup** library in Python. Let's use an example of extracting quotes from the website "http://quotes.toscrape.com/".

- **Step 1: Install Libraries:** Make sure you have the necessary libraries installed. You can install them using pip:

```
pip install requests beautifulsoup4
```

Web Scraping

Beautiful Soup is a Python library that is commonly used for web scraping purposes. It provides tools for parsing HTML and XML documents, navigating their structure, and extracting the data you need. Beautiful Soup makes it easier to work with the complex and nested structure of web pages, allowing you to extract specific information efficiently.
Here are some key features of Beautiful Soup:

1. **Parsing HTML:** Beautiful Soup can parse raw HTML and XML documents and convert them into a structured tree of Python objects. This makes it easy to traverse and manipulate the document's elements and content.
2. **Traversal:** You can navigate through the parsed document using methods like .find() and .find_all() to locate specific elements based on tags, attributes, or other criteria.
3. **Accessing Attributes and Text:** Beautiful Soup provides methods to access the attributes and text content of elements. You can use .get() to access attribute values and .text to access the text content of an element.
4. **Modifying Documents:** You can modify the parsed document by adding, modifying, or removing elements, attributes, and content.
5. **Searching and Filtering:** Beautiful Soup allows you to search for elements based on various filters, such as tag names, attributes, CSS classes, and text content.

Web Scraping

- **Step 2: Import Libraries:** Import the required libraries in your Python script.

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

- **Step 3: Send a Request:** Send an HTTP GET request to the website's URL using the **requests** library.

```
url = "http://quotes.toscrape.com/"
response = requests.get(url)
```

Web Scraping

- **Step 4: Parse HTML:** Parse the HTML content of the response using BeautifulSoup.

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

- **Step 5: Locate and Extract Data:** Use soup to locate and extract the data you're interested in. In this case, we'll extract quotes.

```
quotes = soup.find_all("span", class_="text")
for quote in quotes:
    print(quote.get_text())
```

Web Scraping

Step 6: Pagination (Optional): If the website has multiple pages, you can implement pagination to scrape data from all pages.

```
next_button = soup.find("li", class_="next")
while next_button:
    next_page = url + next_button.a["href"]
    response = requests.get(next_page)
    soup = BeautifulSoup(response.content, "html.parser")

    quotes = soup.find_all("span", class_="text")
    for quote in quotes:
        print(quote.get_text())

    next_button = soup.find("li", class_="next")
```

Web Scraping

Step 6: Pagination (Optional): If the website has multiple pages, you can implement pagination to scrape data from all pages.

```
next_button = soup.find("li", class_="next")
while next_button:
    next_page = url + next_button.a["href"]
    response = requests.get(next_page)
    soup = BeautifulSoup(response.content, "html.parser")

    quotes = soup.find_all("span", class_="text")
    for quote in quotes:
        print(quote.get_text())

    next_button = soup.find("li", class_="next")
```

Go to NoteBook

NeuroTech

Any Question

NeuroTech

Thanks

NeuroTech