Nifty-50 Page Web-Scraping Documentation

**Overview**

This script is designed to perform two primary functions:

1. **Extract Specific HTML Elements**: Extract text from specified HTML tags and save the data into a JSON file.
2. **Extract All HTML Tables**: Extract tables from the webpage and save each table into a separate CSV file.
3. **Url**: <https://groww.in/indices/nifty>

**Workflow**

1. **Import Libraries**: Import essential Python libraries for HTTP requests, HTML parsing, data handling, and file system operations.
2. **Extract HTML Elements**:
   * **Input**: The URL of the webpage, a list of HTML tags to extract, and the filename for the JSON output.
   * **Process**:
     + Fetch the HTML content from the webpage.
     + Parse the HTML to find specified tags.
     + Extract text content from these tags.
     + Save the extracted data into a JSON file.
   * **Output**: A JSON file containing the text content of the specified HTML elements.
3. **Extract HTML Tables**:
   * **Input**: The URL of the webpage from which tables will be extracted.
   * **Process**:
     + Fetch the HTML content of the webpage.
     + Parse the HTML to locate all <table> elements.
     + Extract headers and rows from each table.
     + Save each table’s data into a separate CSV file.
   * **Output**: Multiple CSV files, each containing the data from one table.
4. **File and Directory Management**:
   * **Process**:
     + Create directories for saving JSON and CSV files if they do not exist.
     + Generate filenames based on the URL and ensure unique names for each file.
   * **Output**: Organized files and directories with the scraped data

**Code Explanation**:

* **Imports**:
  + requests: For making HTTP requests to retrieve webpage content.
  + BeautifulSoup: For parsing HTML and extracting data.
  + json: For handling JSON data.
  + csv: For creating and writing to CSV files.
  + re: For regular expression operations.
  + os: For file and directory operations.

**extract\_elements(soup, tags)**

* **Purpose**: To extract and clean text content from specified HTML tags.
* **Parameters**:
  + soup: A BeautifulSoup object with parsed HTML content.
  + tags: A list of HTML tags to target.
* **Returns**: A dictionary with tags as keys and lists of extracted text as values.

**save\_to\_json(data, filename, output\_dir)**

* **Purpose**: To save extracted data to a JSON file.
* **Parameters**:
  + data: The dictionary of extracted data.
  + filename: The name of the JSON file.
  + output\_dir: The directory where the file will be saved.
* **Process**: Creates the output directory if necessary and writes the data to a JSON file with readable formatting.

**scrape\_website(url, tags, output\_filename, output\_dir)**

* **Purpose**: To extract HTML elements from a webpage and save them in JSON format.
* **Parameters**:
  + url: The URL of the webpage to scrape.
  + tags: A list of HTML tags to extract.
  + output\_filename: The name of the JSON file.
  + output\_dir: The directory where the JSON file will be stored.
* **Process**: Fetches webpage content, parses HTML, extracts specified elements, and saves them to a JSON file.

**get\_first\_word\_of\_url(url)**

* **Purpose**: To generate a prefix for output filenames based on the URL.
* **Parameters**:
  + url: The URL of the webpage.
* **Returns**: The first segment of the domain from the URL, used for the filename prefix.

**scrape\_all\_tables(url)**

* **Purpose**: To extract all tables from a webpage and save each table as a CSV file.
* **Parameters**:
  + url: The URL of the webpage to scrape.
* **Process**:
  + **Fetch Webpage Content**: Make an HTTP GET request to the URL.
  + **Parse HTML**: Find all <table> elements using BeautifulSoup.
  + **Extract Data**:
    - **Headers**: Get the text from <th> elements.
    - **Rows**: Get the text from <td> elements.
  + **Save to CSV**: Write each table’s data to a CSV file, naming them sequentially (table\_1.csv, table\_2.csv, etc.).

**Obstacles Faced and Overcome:**

1. **Creating a Regular Expression for Domain Extraction**:
   * **Obstacle**: Formulating a regular expression to extract the first segment of the domain from a URL was challenging.
   * **Solution**: Utilized online AI tools and resources to refine the regular expression, ensuring accurate extraction of the domain prefix.
2. **Handling CSV Files**:
   * **Obstacle**: Limited prior experience with CSV files, as previous work was primarily with JSON files.
   * **Solution**: Referenced official documentation and leveraged AI tools to understand and implement CSV file operations effectively.
3. **Overcoming UnicodeEncodeError**:
   * **Obstacle**: Encountered UnicodeEncodeError when trying to write currency symbols to a CSV file.
   * **Solution**: Conducted research online to identify the appropriate encoding format (UTF-8) for handling special characters, ensuring correct representation in the CSV files.