VakilDesk
LEGAL MANAGEMENT MADE SIMPLER

# Interview Assignment
# Web Scraping using Python

## Overview

The program is designed for web scraping various pages from the website "https://www.scrapethissite.com/". It fetches data from different types of pages containing databases and stores the scraped data into a MongoDB database. The program is equipped with error handling, logging, and parallel scraping to enhance efficiency and robustness.

### Dependencies:

- **requests**: For making HTTP requests to the website.
- **beautifulsoup4**: For parsing HTML content. It helps in extracting data from web pages.
- **pymongo**: For interacting with MongoDB.
- **logging**: For error logging. It allows the script to connect to a MongoDB database and perform operations such as inserting data.
- **Concurrent.futures**: Specifically, **ThreadPoolExecutor** is used for parallel execution of web scraping tasks to speed up the process.

## Setup and Installation

1. Ensure that all the python libraries mentioned above are installed.
2. Ensure MongoDB is installed and running. The project connects to MongoDB at **mongodb://localhost:27017/.**

## Explanation of the Code

### Functions used

1. **get_text_or_none(cell)** :- Extracts and returns text from an HTML cell, or None if the cell is empty.

2. **get_int_value(cell)** :- Converts text from an HTML cell to an integer, returns None if conversion fails.
3. **get_float_value(cell) :** Extracts a float value from an HTML cell, with error handling for conversion issues
4. **different_collections(url) :** Returns a list of year collections from the URL by extracting text from links with the class year-link.
5. **ajax_data(url, year) :** Fetches Ajax data for a given year from the URL.

   **Steps:**

   - Construct the URL with query parameters (ajax=true and year=<year>) to request data for the specified year.
   - Send an HTTP GET request to the constructed URL.
   - Check if the response status code is 200 to ensure a successful request.
   - If successful, parse the JSON response and return the data.
   - If the request fails or the response status code is not 200, log an error message and return None.

6. **parse_team_data(url):** Parses team data from the URL, extracting relevant fields from the HTML content.

   **Steps:**

   - Send an HTTP GET request to the specified URL to retrieve the HTML content of the page.
   - Parse the HTML content using BeautifulSoup to navigate and extract data.
   - Find all the table rows (<tr>) with the class "team", which typically represent individual teams' data rows.
   - Iterate over each team row and extract relevant information such as team name, year, wins, losses, etc.
   - Construct a dictionary for each team with the extracted information.
   - Append each team dictionary to a list of teams.
   - Return the list of teams.

7. **get_total_pages(url) :** Returns the total number of pages available for pagination.
8. **scrape_all_pages(base_url):** Scrapes all pages using parallel scraping, minimizing load on the server with delays.

   **Steps**

   - Obtain the total number of pages to scrape from the base URL.
   - Set up a ThreadPoolExecutor with a defined maximum number of worker threads.
   - Generate futures for each page, representing the asynchronous scraping task.
   - Submit the futures to the executor for simultaneous execution.
   - Process completed futures, retrieving their results and aggregating the team data.
   - Implement error handling to catch and log any exceptions that occur during future execution.
   - Once all pages are processed, return the aggregated team data for further use.

9. **advanced_topic(advanced_url) :** Fetches advanced topic data from the URL.

   **Steps:**

   - Send an HTTP GET request to the specified URL to retrieve the HTML content of the page.
   - Parse the HTML content using BeautifulSoup to navigate and extract data.
   - Identify relevant elements containing advanced topics, typically <h4> headings.
   - For each advanced topic element, extract the topic name, link, and related paragraph text.
   - Construct a dictionary for each advanced topic with the extracted information.
   - Append each topic dictionary to a list of topics.
   - Return the list of advanced topics.

10. **save_to_mongo(collection, data):** Saves the scraped data to MongoDB.
11. **scrape_and_save(url, db_name, collection_name, data_func, *args):** Main function to scrape data and save it to MongoDB.

# Main Scraping Logic

## AJAX JavaScript Page

1. URL: https://www.scrapethissite.com/pages/ajax-javascript/#2015
2. Database: Oscar_Winning_Films
3. Process:
    - Fetch different year collections using different_collections.
    - For each year, scrape AJAX data and save it to MongoDB using scrape_and_save. ach year's data is stored as a separate collection inside the main database, with each collection representing data for a specific year.

## Forms Page

1. URL: https://www.scrapethissite.com/pages/forms/
2. Database: Hockey_Teams_Data
3. Process:
    - Scrape all team data from paginated pages using scrape_all_pages.
    - Save the data to MongoDB using scrape_and_save.

# Advanced Topics Page

1. URL: https://www.scrapethissite.com/pages/advanced/
2. Database: Advanced_topics
3. Process:
    - Scrape advanced topics data using advanced_topic.
    - Save the data to MongoDB using scrape_and_save.

## Execution Time

The script also records and prints the total execution time to measure performance using the time library.