# Web Scraping Documentation

## 1. Introduction

This document outlines the procedure for extracting key job details from job listings on the Gaapweb website. It includes information on how to extract data, handle pagination, run the script in headless mode, display the extracted data as a DataFrame, and adhere to best practices while scraping.

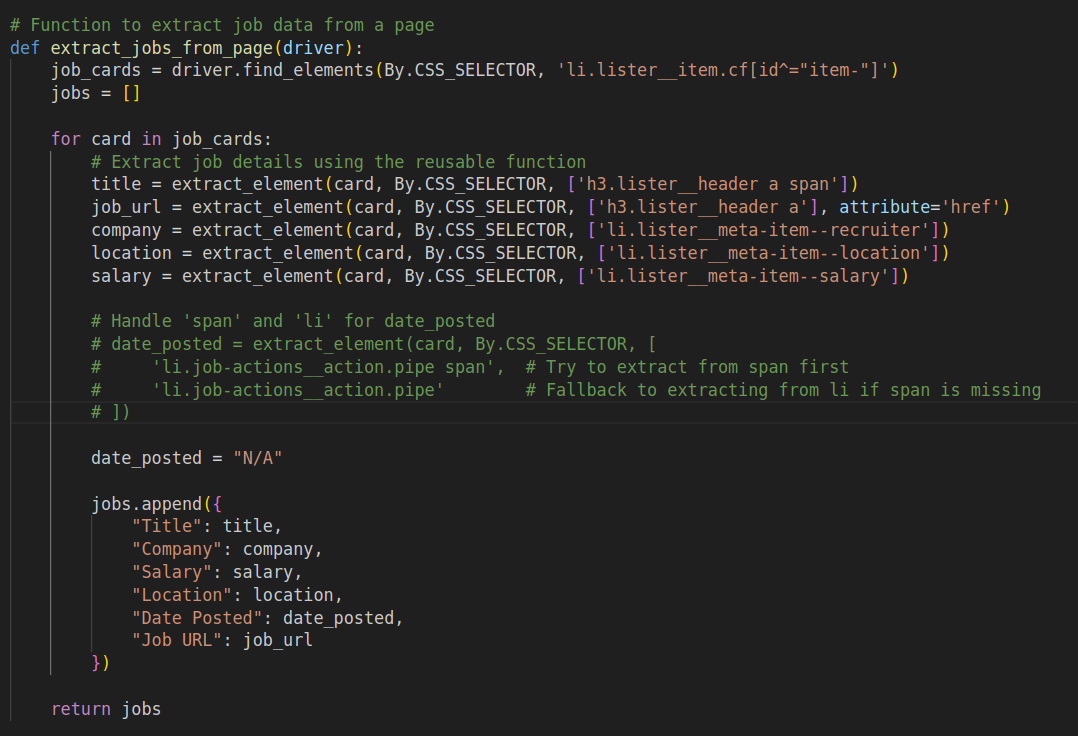
## 2. Procedure for Extracting Key Details

### 2.1 Extracting Key Job Details

The following details are extracted from each job listing:

* Job Title: The title of the job position.
* Company Name: The name of the company offering the job.
* Location: The geographical location of the job.
* Salary: The offered salary, if available.
* Date Posted: The time elapsed since the job was posted, formatted as a date.
* Job URL: The link to the job listing for more information.(not founded in page)

To extract these details, Selenium is used to locate the relevant elements in the HTML structure of the webpage. Below is the relevant Python code snippet:



### 2.2 Handling Pagination

To scrape multiple pages of job listings, the script navigates to the next page by finding the 'Next' button and clicking it until there are no more pages left. The following code snippet demonstrates this approach:

### 2.3 Running the Script in Headless Mode

Headless mode allows the script to run without opening a browser window, which can improve performance and make it suitable for running on servers. To enable headless mode, the following line should be uncommented in the options setup:

### 2.4 Displaying Extracted Data as a DataFrame

The extracted job data can be displayed as a Pandas DataFrame for easier analysis and manipulation. The following code snippet demonstrates how to convert the job list to a DataFrame and print it:

import pandas as pd  
  
df\_jobs = pd.DataFrame(all\_jobs)  
print(df\_jobs)

## 3. Best Practices While Scraping

* Respect robots.txt: Always check the website's robots.txt file to ensure scraping is permitted.
* Rate Limiting: Implement random delays between requests to avoid overwhelming the server. This mimics human behavior and reduces the risk of being blocked.
* Error Handling: Use try-except blocks to handle exceptions gracefully and log errors for debugging.
* User-Agent Rotation: Consider rotating user-agent strings to prevent detection as a bot.