Data Scraping

Web Scraping

It’s the process of automatically gathering information from websites on the internet. Instead of manually copying and pasting data, web scraping allows you to collect large amounts of data quickly and efficiently using automated tools.

Uses of Web Scraping

1. **Financial Data Analysis:**

   Helps collect data like stock prices, financial reports, or economic indicators to analyze market trends and make informed decisions.

2. **Marketing and Sales:**

   Companies use web scraping to gather information about customer preferences, product reviews, and competitor pricing, helping them tailor their marketing strategies.

3. **Academic Research:**

   Researchers scrape data from websites to gather large datasets for studies, experiments, or reports.

4. **Journalism:**

   Journalists can use web scraping to collect data for investigative stories, such as tracking political donations, analyzing trends in public records, or monitoring social media activity.

5. **Real Estate:**

   Real estate professionals use scraping to collect data on property prices, rental rates, and market trends from various real estate websites.

6. **Machine Learning:**

   Web scraping is often used to gather training data for machine learning models, such as collecting images, text, or other data types to improve algorithms.

7. **Brand Monitoring and Competition Analysis:**

   Companies monitor mentions of their brand across the web or track their competitors’ activities, including product launches, pricing changes, and customer feedback.

8. **Social Media Analysis:**

   Web scraping is used to gather data from social media platforms to analyze trends, sentiments, and user engagement.

Web Crawling vs. Web Scraping

Web Crawling:

- **What it is:** Web crawling is the process of automatically scanning and indexing information on web pages using special bots called crawlers.
- **How it works:** Crawlers browse through large numbers of websites, downloading and storing their content. This is usually done on a large scale, like the way search engines gather information.
- **Purpose:** It's used to gather generic information from websites, such as URLs, metadata, and links.
- **Examples:** Google, Bing, and Yahoo use web crawlers (like Googlebot) to index websites for their search engines.

Web Scraping:

- **What it is:** Web scraping is the automated process of extracting specific pieces of data from websites using bots called scrapers.
- **How it works:** Scrapers are designed to target specific data elements from a website, such as prices, reviews, or contact information, using the structure of the site.
- **Purpose:** Web scraping can be done on any scale and is used to collect detailed, specific information for purposes like data analysis.
- **Use Cases:** While web crawling collects broad information, web scraping focuses on precise data, which can be used for various applications, including research, business intelligence, and more.

Components of a Web Scraper

1. **Web Crawler Module:**

   The web crawler module is responsible for navigating the web and finding the pages that need to be scraped.

2. **Extractor (Parser):**

   Once the pages are found, the extractor (or parser) is used to pull out the specific pieces of data you need. It scans the HTML structure of the page and identifies the exact information to extract.

3. **Data Transformation and Cleaning Module:**

   After data is extracted, this module cleans and formats it into a usable form. It handles issues like missing data, duplicates, or formatting inconsistencies, ensuring the data is ready for analysis or storage.

4. **Storage Module:**

   Finally, the storage module saves the cleaned and transformed data. This could be in a database, a spreadsheet, or any other form that's easy to access and use later.

How a Web Scraper Works

- **Step 1:** The web crawler module starts by visiting a website or a list of URLs.
- **Step 2:** The crawler finds the relevant web pages and passes them to the extractor.
- **Step 3:** The extractor identifies and pulls out the specific data elements from these pages.

- **Step 4:** The extracted data goes through the data transformation and cleaning module to ensure it's accurate and consistent.
- **Step 5:** The cleaned data is stored in the storage module, ready for analysis or other uses.

Challenges of Web Scraping

1. **Variety:**
    - **What it means:** Websites come in all shapes and sizes, with different structures, formats, and content types. This variety can make it challenging to create a one-size-fits-all scraper.
    - **Why it's a challenge:** Each website might require a different approach to extract the needed data, requiring more complex or adaptable scraping tools.
2. **Durability:**
    - **What it means:** Websites often change their layout, structure, or content, which can break the scraper or make it less effective.
    - **Why it's a challenge:** Web scrapers need to be maintained and updated regularly to keep up with these changes, ensuring they continue to work correctly.

Alternative to Web Scraping

Application Programming Interfaces (APIs):

What are APIs? Some websites offer APIs, which are tools that allow you to access their data in a structured and predefined way. Instead of scraping the entire web page, you can directly request the specific data you need through the API.

Why use APIs?

- **Simpler Data Access:** APIs provide a more straightforward way to get data without needing to parse through complex HTML structures.
- **Formats:** Data is usually provided in easy-to-use formats like JSON or XML, making it much easier to work with compared to raw HTML.
- **Efficiency:** Using APIs can be more efficient and reliable than web scraping since the data is directly provided by the website in a clean and organized manner.

Static Web Scraping

- A static website is one where the content is fixed and doesn't change dynamically based on user interactions. The content is the same for every visitor and doesn't require any server-side processing to generate the web page.
- **Example:** A basic HTML webpage that shows the same text, images, and layout every time you visit it.

Steps:

1. **Inspect Your Data Source:**

   Start by examining the website from which you want to scrape data. Look at the structure of the webpage to understand where the data is located.

   - **Explore the Website:** Navigate through the website to see how it works. Check different pages and links to understand how the data is organized.
   - **Decrypt the URL:** Sometimes, URLs can be complicated or hidden. Make sure you find the correct URLs that point to the data you need.
   - **Use Developer Tools:** Use your browser's developer tools (like Chrome DevTools) to inspect elements on the webpage. This helps you identify the specific HTML tags and classes where the data is stored.

2. **Scrape Content:**

   Requests is a powerful Python library used to send HTTP requests to the website to get the page content. By sending a **GET** request to a URL, you can retrieve the HTML of the webpage, which you will later parse to extract data.

3. **Parse Content:**

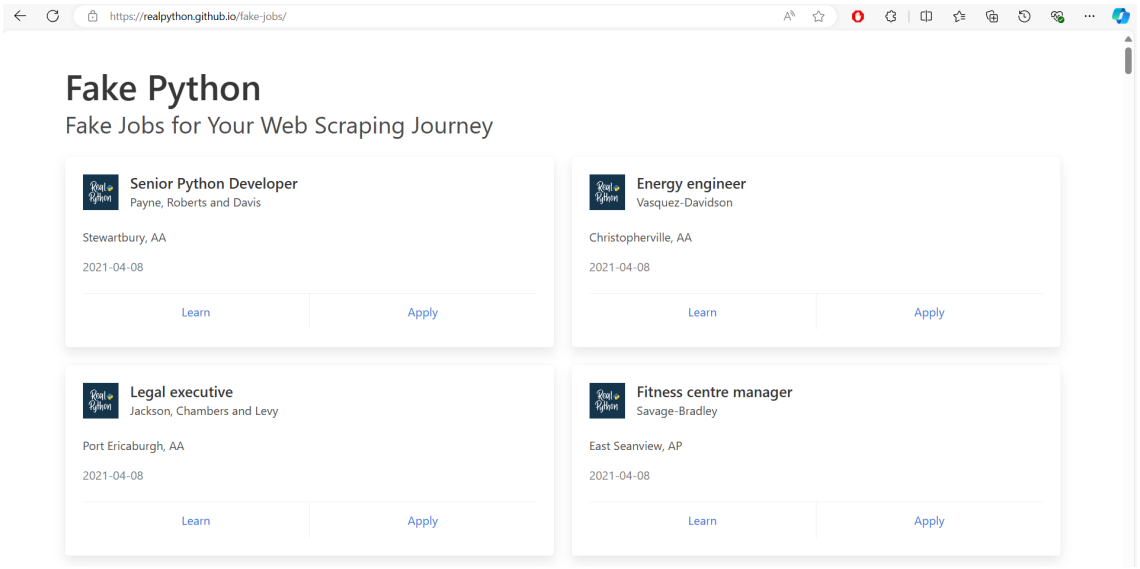
   After getting the HTML content, parse it to find and extract the specific data elements you need. **BeautifulSoup** is a Python library that allows you to navigate the HTML structure and select specific elements, tags, or classes to extract data.

Code in Python

Step 1:

```
# Import Libraries

import requests
from bs4 import BeautifulSoup
import re
```

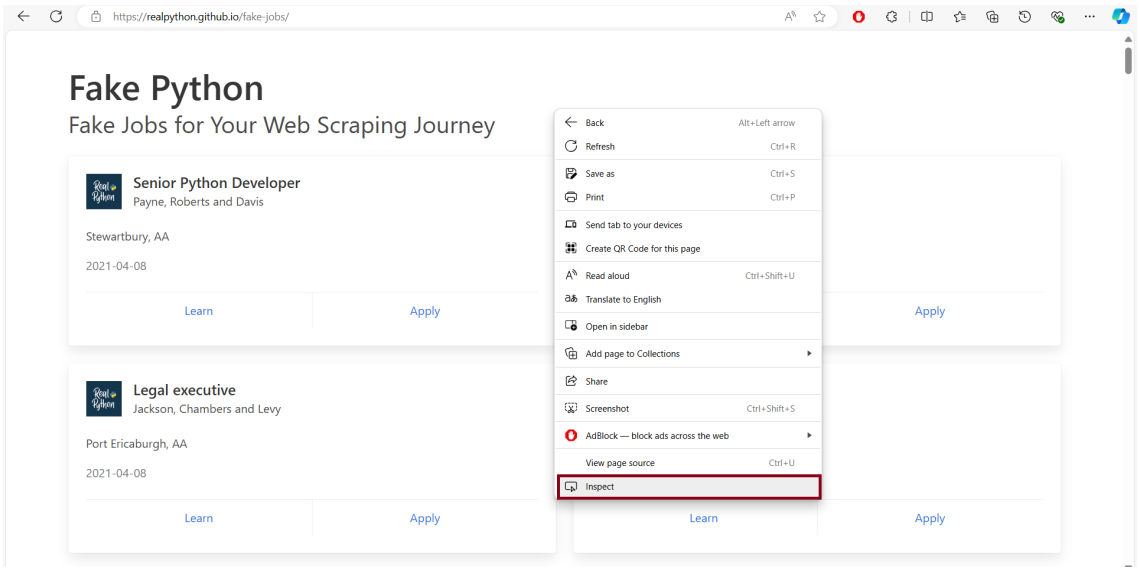
Step 2:

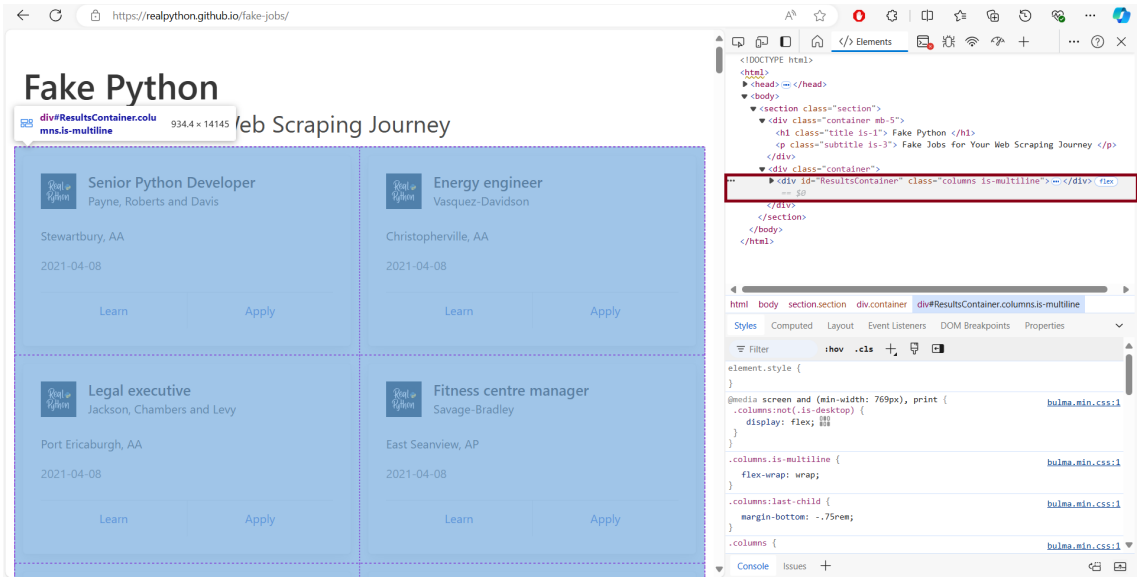



```
# Define the URL of the webpage to scrape
URL = "https://realpython.github.io/fake-jobs/"

# Send a GET request to the specified URL and store the response in 'page'
page = requests.get(URL)

# Print the raw HTML content of the page as text
print(page.text)
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Fake Python</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.2/css/bulma.min.css">
  </head>
  <body>
  <section class="section">
    <div class="container mb-5">
      <h1 class="title is-1">
        Fake Python
      </h1>
      <p class="subtitle is-3">
        Fake Jobs for Your Web Scraping Journey
      </p>
    </div>
    <div class="container">
    <div id="ResultsContainer" class="columns is-multiline">
    <div class="column is-half">
```

Step 3:

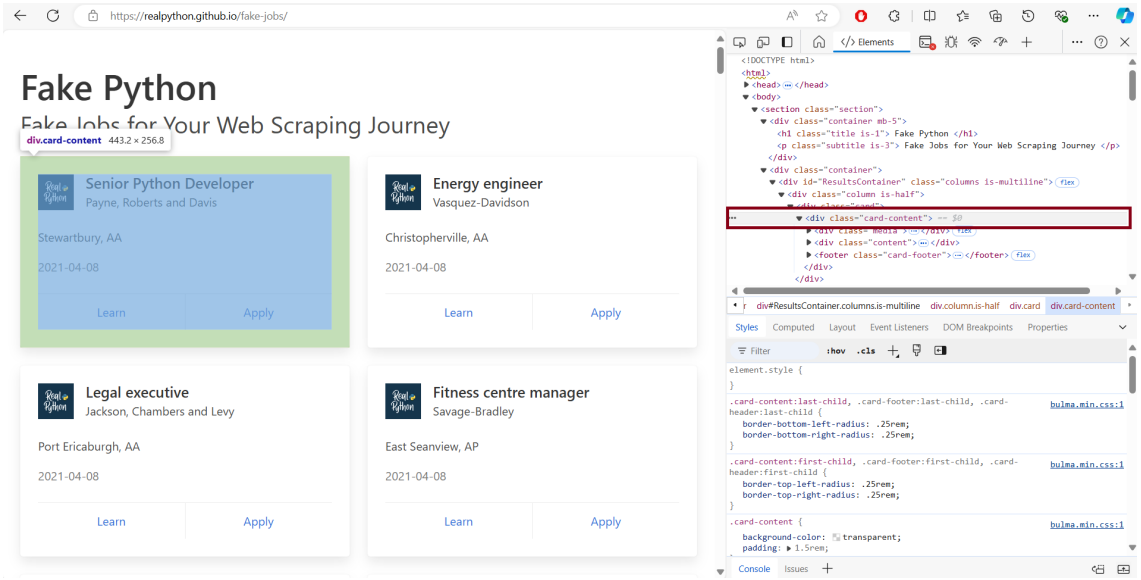





```
# Parse the page content using BeautifulSoup, specifying the HTML parser
soup = BeautifulSoup(page.content, "html.parser")

# Find the HTML element with the id 'ResultsContainer' and store it in the varia
results = soup.find(id="ResultsContainer")

# Print the HTML content of the 'results' variable in a formatted (pretty) way
print(results.prettify())
```

```
<div class="columns is-multiline" id="ResultsContainer">
 <div class="column is-half">
  <div class="card">
   <div class="card-content">
    <div class="media">
     <div class="media-left">
      <figure class="image is-48x48">
       <img alt="Real Python Logo" src="https://files.realpython.com/media/real-python-logo-thumbnail.7f0db70c2ed2.jpg?__no_cf_polish=1"/>
      </figure>
     </div>
     <div class="media-content">
      <h2 class="title is-5">
       Senior Python Developer
      </h2>
      <h3 class="subtitle is-6 company">
       Payne, Roberts and Davis
      </h3>
     </div>
    </div>
    <div class="content">
     <p class="location">
      Stewartbury, AA
     </p>
     <p class="is-small has-text-grey">
```

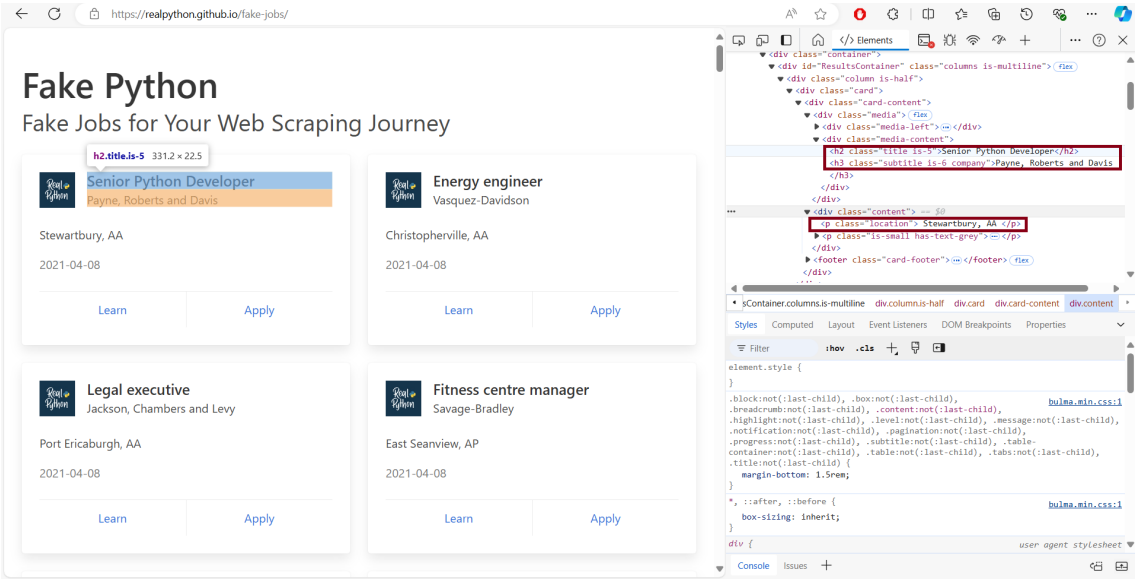
Step 4:




```
# Find all job elements with the class 'card-content' and store them in 'job_ele
job_elements = results.find_all("div", class_="card-content")

# Print each job element with extra spacing for readability
for job_element in job_elements:
    print(job_element, end="\n"*2)
```

```
<div class="card-content">
<div class="media">
<div class="media-left">
<figure class="image is-48x48">
<img alt="Real Python Logo" src="https://files.realpython.com/media/real-python-logo-thumbnail.7f0db70c2ed2.jpg?__no_cf_polish=1"/>
</figure>
</div>
<div class="media-content">
<h2 class="title is-5">Senior Python Developer</h2>
<h3 class="subtitle is-6 company">Payne, Roberts and Davis</h3>
</div>
</div>
<div class="content">
<p class="location">
      Stewartbury, AA
    </p>
<p class="is-small has-text-grey">
<time datetime="2021-04-08">2021-04-08</time>
</p>
</div>
<footer class="card-footer">
<a class="card-footer-item" href="https://www.realpython.com" target="_blank">Learn</a>
<a class="card-footer-item" href="https://realpython.github.io/fake-jobs/jobs/senior-python-developer-0.html" target="_blank">Apply</a>
</footer>
</div>

<div class="card-content">
<div class="media">
```

Step 5:




```
# Extract and print the title, company, and location from each job element
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element)
    print(company_element)
    print(location_element)
    print()
```

```
<h2 class="title is-5">Senior Python Developer</h2>
<h3 class="subtitle is-6 company">Payne, Roberts and Davis</h3>
<p class="location">
        Stewartbury, AA
      </p>

<h2 class="title is-5">Energy engineer</h2>
<h3 class="subtitle is-6 company">Vasquez-Davidson</h3>
<p class="location">
        Christopherville, AA
      </p>

<h2 class="title is-5">Legal executive</h2>
<h3 class="subtitle is-6 company">Jackson, Chambers and Levy</h3>
<p class="location">
        Port Ericaburgh, AA
      </p>
```

```
# Extract and print the text of title, company, and location from each job eleme
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text)
    print(company_element.text)
    print(location_element.text)
    print()
```

```
Senior Python Developer
Payne, Roberts and Davis

        Stewartbury, AA


Energy engineer
Vasquez-Davidson

        Christopherville, AA


Legal executive
Jackson, Chambers and Levy

        Port Ericaburgh, AA
```

```
# Extract and print the stripped text of title, company, and location from each
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
```

```
    location_element = job_element.find("p", class_="location")
    print(title_element.text.strip())
    print(company_element.text.strip())
    print(location_element.text.strip())
    print()
```

```
Senior Python Developer
Payne, Roberts and Davis
Stewartbury, AA

Energy engineer
Vasquez-Davidson
Christopherville, AA

Legal executive
Jackson, Chambers and Levy
Port Ericaburgh, AA
```

Step 6:

```
# Find all job titles containing the word "Python" and store them in 'python_job
python_jobs = results.find_all("h2", string=re.compile("Python"))

# Locate the parent job elements for job titles containing "Python"
python_job_elements = [
    h2_element.parent.parent.parent for h2_element in python_jobs
]

# Extract and print the title, company, and location for the located parent job
for job_element in python_job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text.strip())
    print(company_element.text.strip())
    print(location_element.text.strip())
    print()
```

```
Senior Python Developer
Payne, Roberts and Davis
Stewartbury, AA

Software Engineer (Python)
Garcia PLC
Ericberg, AE

Python Programmer (Entry-Level)
Moss, Duncan and Allen
Port Sara, AE

Python Programmer (Entry-Level)
Cooper and Sons
West Victor, AE
```

Step 7:

```
# For each job element in 'python_job_elements', find all 'a' tags (links)
for job_element in python_job_elements:
    links = job_element.find_all("a")
    # Print the URL of each link
    for link in links:
        link_url = link["href"]
        print(f"Link: {link_url}\n")
```

```
Link: https://www.realpython.com

Link: https://realpython.github.io/fake-jobs/jobs/senior-python-developer-0.html

Link: https://www.realpython.com

Link: https://realpython.github.io/fake-jobs/jobs/software-engineer-python-10.html

Link: https://www.realpython.com

Link: https://realpython.github.io/fake-jobs/jobs/python-programmer-entry-level-20.html
```

```
# For each job element in 'python_job_elements', print the URL of the second 'a'
for job_element in python_job_elements:
    # Access the second 'a' tag and print its 'href' attribute as the applicatio
    link_url = job_element.find_all("a")[1]["href"]
    print(f"Apply here: {link_url}\n")
```

```
Apply here: https://realpython.github.io/fake-jobs/jobs/senior-python-developer-0.html

Apply here: https://realpython.github.io/fake-jobs/jobs/software-engineer-python-10.html

Apply here: https://realpython.github.io/fake-jobs/jobs/python-programmer-entry-level-20.html
```

Dynamic Web Scraping

- A dynamic website is one where the content changes based on user interactions or other factors, like scrolling or clicking. The content is often generated on the server side and may not be visible in the HTML source code until after the page has loaded or the user has interacted with it.
- **Example:** A webpage that loads additional content as you scroll down or a site that requires you to click a button to see more information.

Steps:

1. **Inspect Your Data Source:**

   Begin by understanding how the dynamic content is loaded on the website. This often involves interacting with the page to see how the content changes.

   - **Explore the Website:** Navigate the site to observe when and how data appears. Notice if content loads after scrolling or clicking.
   - **Decrypt the URL:** Dynamic sites might change the URL based on user actions. Pay attention to how URLs change as you interact with the site.
   - **Use Developer Tools:** Utilize browser developer tools (like Chrome DevTools) to observe the network activity. This can show you when and where data is being loaded from, even if it isn't visible in the initial HTML source.

2. **Interact with the Webpage:**

   Selenium is a Python tool that automates web browsers. It allows you to simulate user actions like clicking, typing, or scrolling to load dynamic content. This is crucial for scraping data from dynamic websites, where content isn't fully loaded until a user action is performed.

   - **Using WebDriver:** Selenium's WebDriver is used to control the browser. You can open webpages, click buttons, and even scroll through pages to load all the content.
   - **Simulating User Actions:** For example, if the data loads after clicking a "Load More" button, you would use Selenium to click that button before attempting to scrape the content.

3. **Scrape Content:**

   Once the dynamic content is loaded, use Selenium to capture the HTML of the webpage. Then, you can pass this HTML to **BeautifulSoup** to parse and extract the specific data elements you need.

   - **Extract Outer HTML:** Use `house.get_attribute("outerHTML")` to get the HTML of specific dynamic elements on the page.
   - **Parse with BeautifulSoup:** After capturing the dynamic content, parse it with BeautifulSoup just like you would with static content. This allows you to locate specific tags and classes to

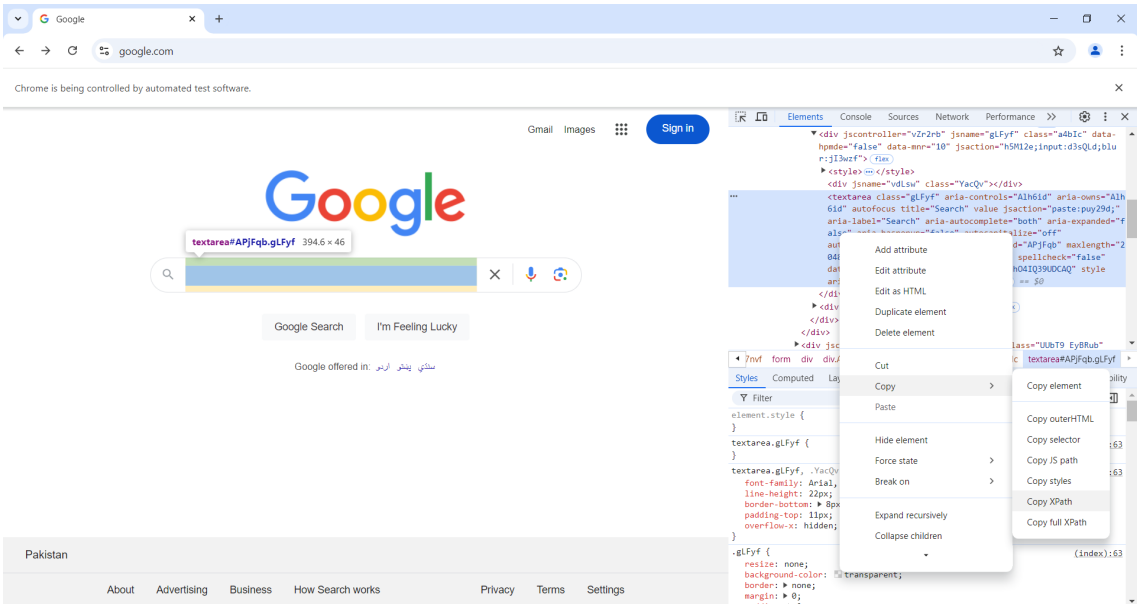
extract the necessary data.

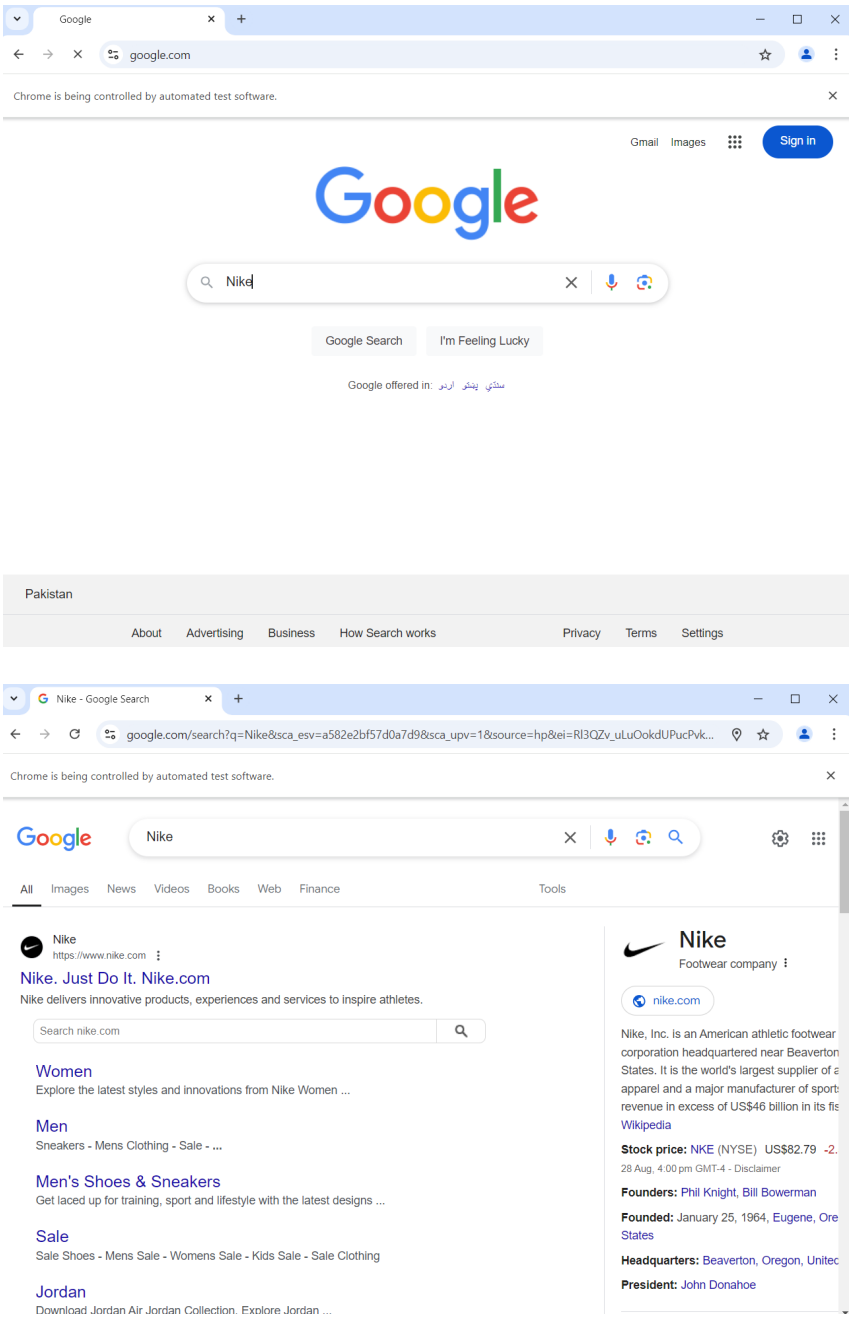
Code in Python

Step 1:

```
from selenium import webdriver
from bs4 import BeautifulSoup
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
import pandas as pd

# Initialize the Chrome WebDriver using the specified path
driver = webdriver.Chrome(service=Service("D:\\Apps\\chromedriver-win64\\chromed
```

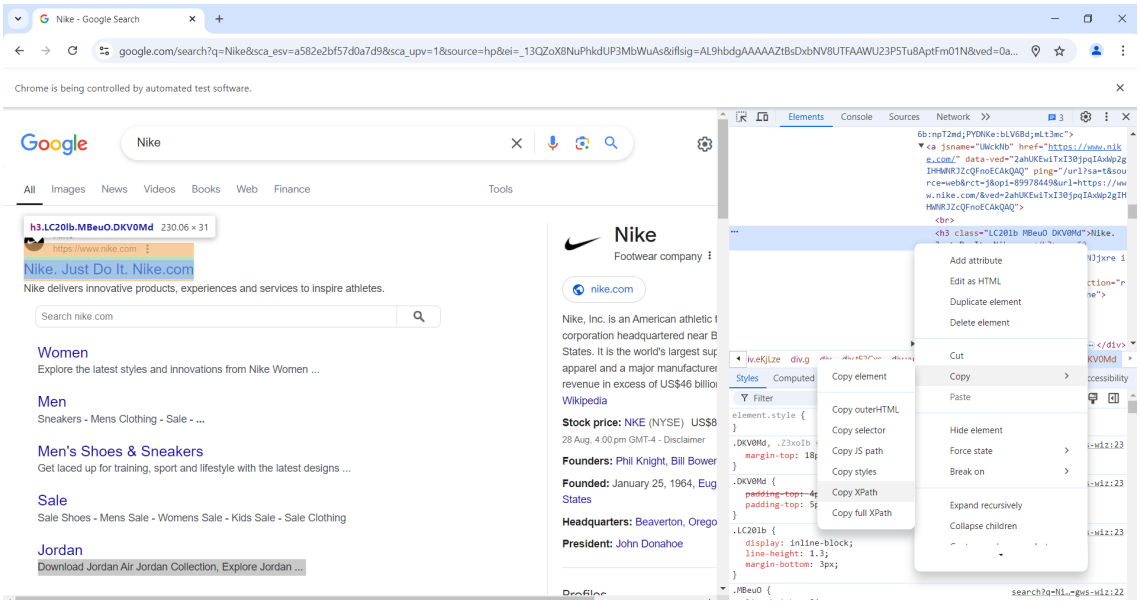
Step 2:

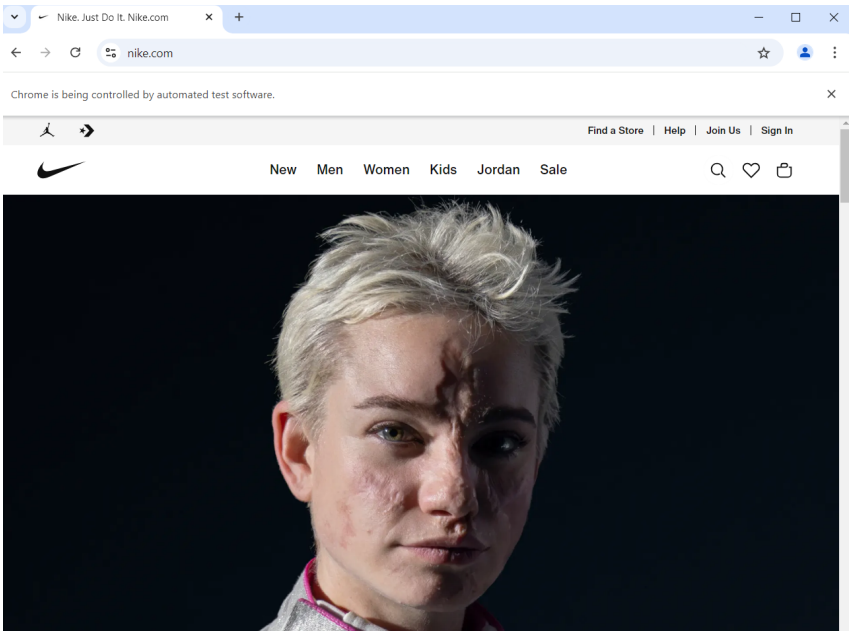





```
# Navigate to Google's homepage
driver.get("https://www.google.com/")
time.sleep(2) # Wait for 2 seconds to ensure the page is fully loaded

# Find the search bar using XPath and enter "Nike" as the search term
search = driver.find_element(by=By.XPATH, value='//*[@id="APjFqb"]')
search.send_keys("Nike")
search.send_keys(Keys.ENTER)
```

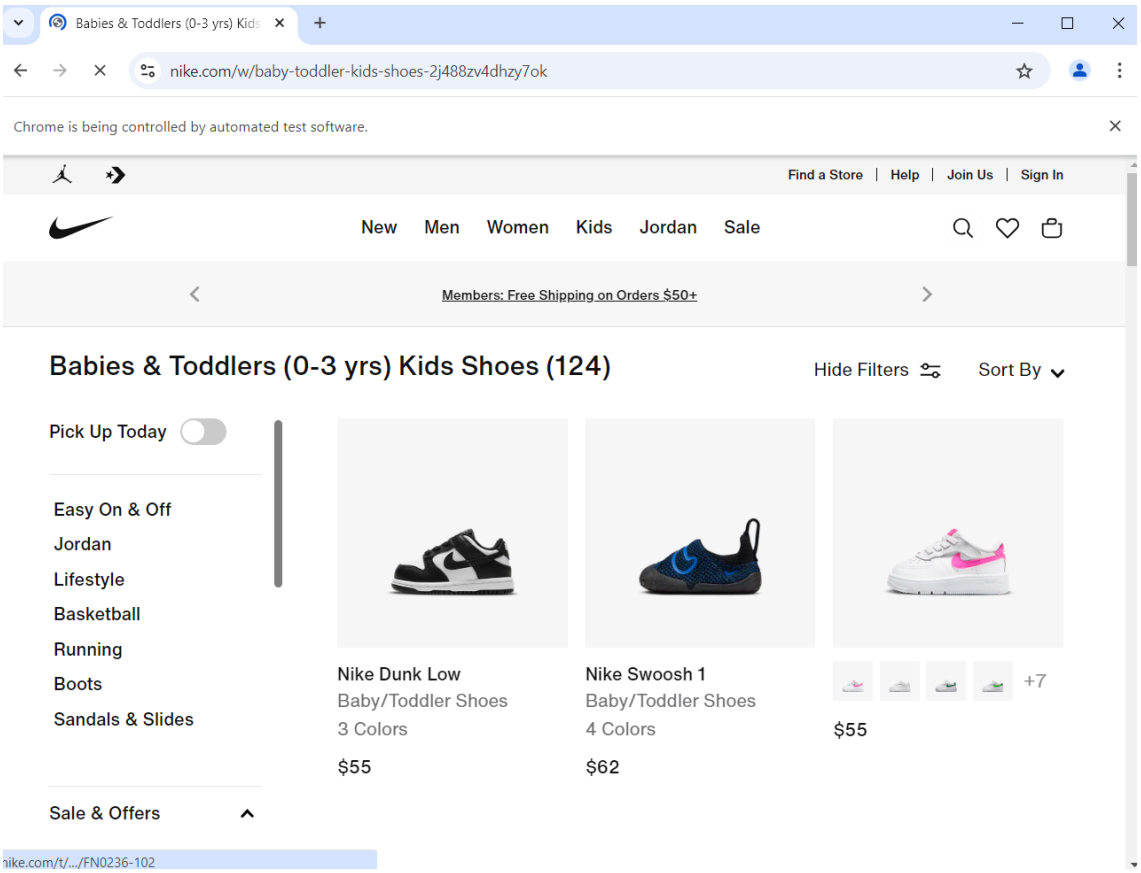
Step 3:






```
# Click on the first search result for Nike's website
driver.find_element(by=By.XPATH, value='/html/body/div[3]/div/div[12]/div[1]/div
```

Step 4:

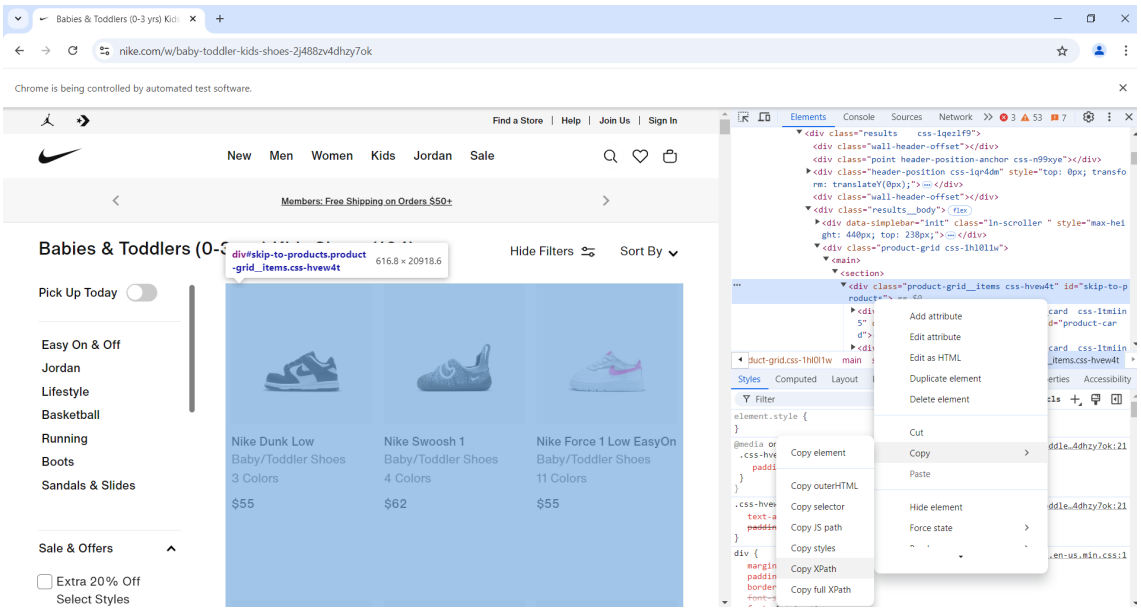


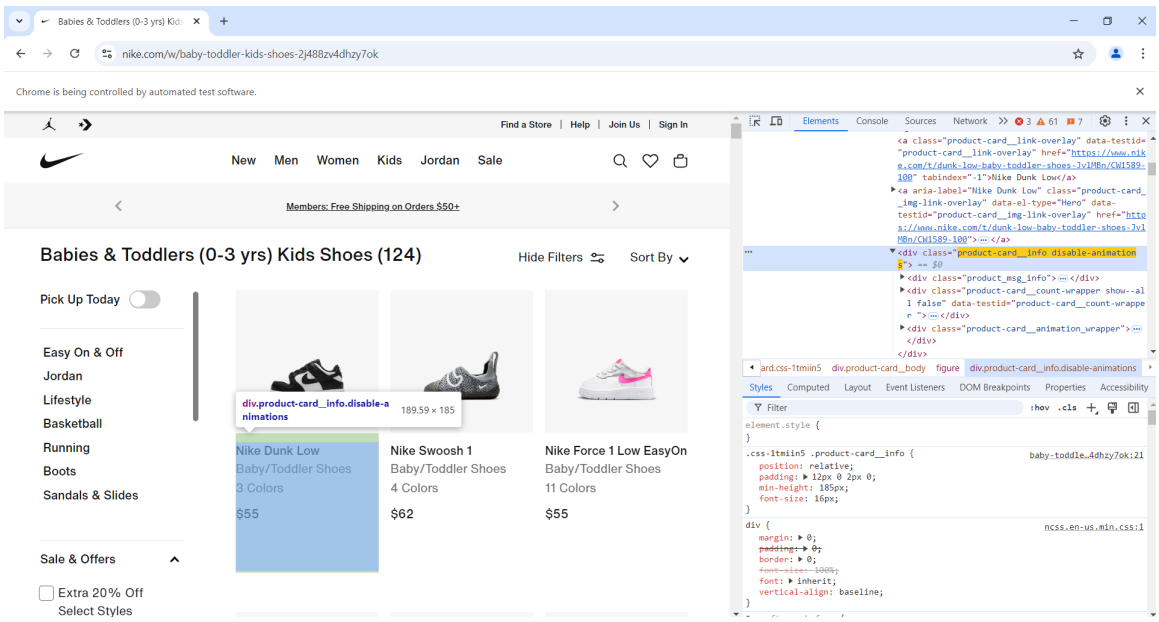

```
# Navigate to the Baby & Toddler Shoes section on Nike's website
driver.get("https://www.nike.com/w/baby-toddler-kids-shoes-2j488zv4dhzy7ok")
```

Step 5:

```
# Scroll down the page until all content is loaded
while True:
    height = driver.execute_script("return document.body.scrollHeight")  # Get t
    driver.execute_script("window.scrollTo(0,document.body.scrollHeight)")  # Sc
    time.sleep(4)  # Wait for 4 seconds to allow new content to load
    h2 = driver.execute_script("return document.body.scrollHeight")  # Get the r

    if height == h2:  # If the scroll height hasn't changed, all content is loac
        break
```

Step 6:






```
# Locate the section containing the product information
content = driver.find_element(by=By.XPATH, value='/html/body/div[4]/div/div/div[

# Parse the HTML content of the product section using BeautifulSoup
html = BeautifulSoup(content.get_attribute("outerHTML"), 'html.parser')
all_products = html.find_all("div", class_="product-card__info disable-animation
```

Step 7:

```
# Initialize lists to store product names, color options, and prices
names = []
colors = []
prices = []

# Loop through each product card and extract the name, color options, and price
for product in all_products:
    name = product.find("div", class_="product-card__title")
    color = product.find("div", class_="product-card__product-count")
    price = product.find("div", class_ = "product-card__price")
    names.append(name.text.strip())
    colors.append(color.text.strip())
    prices.append(price.text.strip())

# Create a DataFrame to organize the extracted data
df2 = pd.DataFrame()
df2["Product Name"] = names
df2["Color Options"] = colors
df2["Price"] = prices

# Display the DataFrame
df2
```

	Product Name	Color Options	Price
0	Nike Dunk Low	3 Colors	$55
1	Nike Swoosh 1	4 Colors	$62
2	Nike Force 1 Low EasyOn	11 Colors	$55
3	Nike Blazer Mid '77	5 Colors	$65
4	Nike Force 1 Mid EasyOn	2 Colors	$60
...	...	...	...
86	Tatum 2	1 Color	$55
87	Nike Blazer Mid '77	1 Color	$65
88	Jordan 1 Mid Wings	1 Color	$65
89	Nike Force 1 Low EasyOn	1 Color	$53.97$65
90	Nike Flex Advance SE	1 Color	$41.97$65

Here are some tags:

- <!DOCTYPE html> declaration defines this document to be HTML5
- <html> element is the root element of an HTML page
- tag defines a division or a section in an HTML document. It's usually a container for other elements
- <head> element contains meta information about the document
- <title> element specifies a title for the document
- <body> element contains the visible page content
- <h1> element defines a large heading
- <p> element defines a paragraph
- <a> element defines a hyperlink
- < table > for tables
- < tr > for table rows
- < th > for table headers
- < td > for table cells