Outline

Tuesday, October 15, 2024 7:47 PM

Pre-requisites:

- Basic knowledge of Python and Pandas
- Basic understanding of how the web works
- Basic knowledge of HTML

Quick Links for Reference:

- Session 1 on Basic working of Web - link
- Session 2 on Data Cleaning - link

Course Structure:

1. **Introduction to Web Scraping**
    - *What is web scraping?*
    - *Ethical Considerations*
    - *Data on the web*
    - *Tools for web scraping*
2. **Setting up Python environment**
    - *Anaconda*
    - *Installing necessary libraries*
    - *Environment setup*
3. **Primer on Web**
    - *How websites interact*

- *HTTP methods*
- *Status codes*

4. **Interacting with the Web using Requests**
5. **HTML parsing using Beautiful Soup**
6. **Web Automation using Selenium**
7. **Case Study: A Complete Web Scraping Project**

Topics

Tuesday, October 15, 2024 7:48 PM

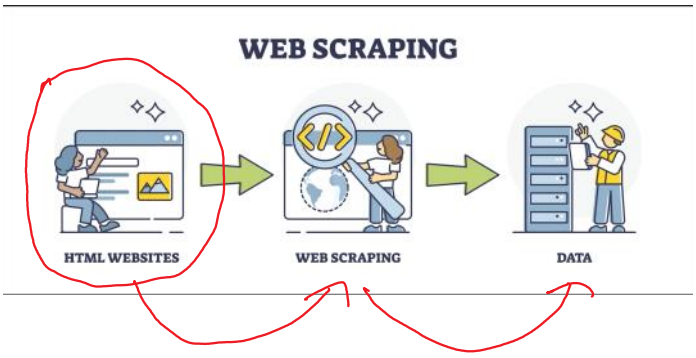
1. **What is Web Scraping?**
2. **Types of Web Scraping**
3. **Ethical Considerations**
4. **Advantages**
5. **Challenges & Disadvantages**
6. **Alternatives to Web Scraping**

Basics

Tuesday, October 29, 2024 10:39 PM

What is Web Scraping?

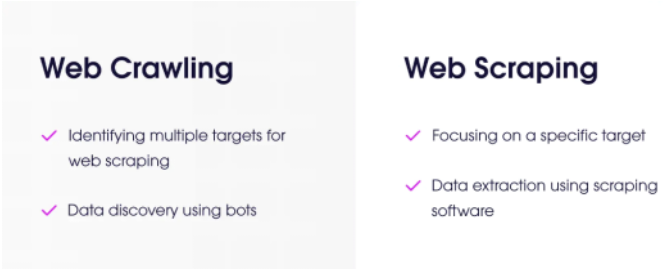
- Web scraping is the automated process of gathering data from websites.
- It's like a bot that navigates through a webpage and collects data based on predefined instructions.
- This data can range from product prices to text content on articles, images, or structured data in tables.



How is Web Scraping different from Web Crawling?

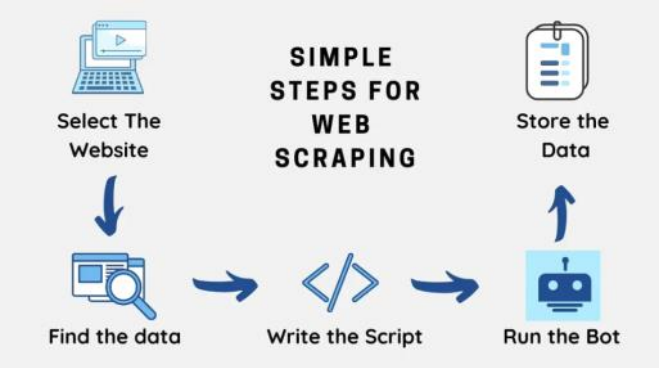
- Web crawling, also known as spidering, is the process of systematically navigating the internet to discover and index web pages.
- Web crawlers (or spiders) start from a set of URLs, visit each page, extract links to other pages, and continue visiting new pages in a recursive manner.
- This enables the crawler to build an extensive index of web pages across a domain or even the entire internet.
- The main goal of web crawling is to find and catalog all accessible pages on the web.
- Crawled pages are often stored in a database or index for later retrieval and use, such as by search engines or content aggregation tools.

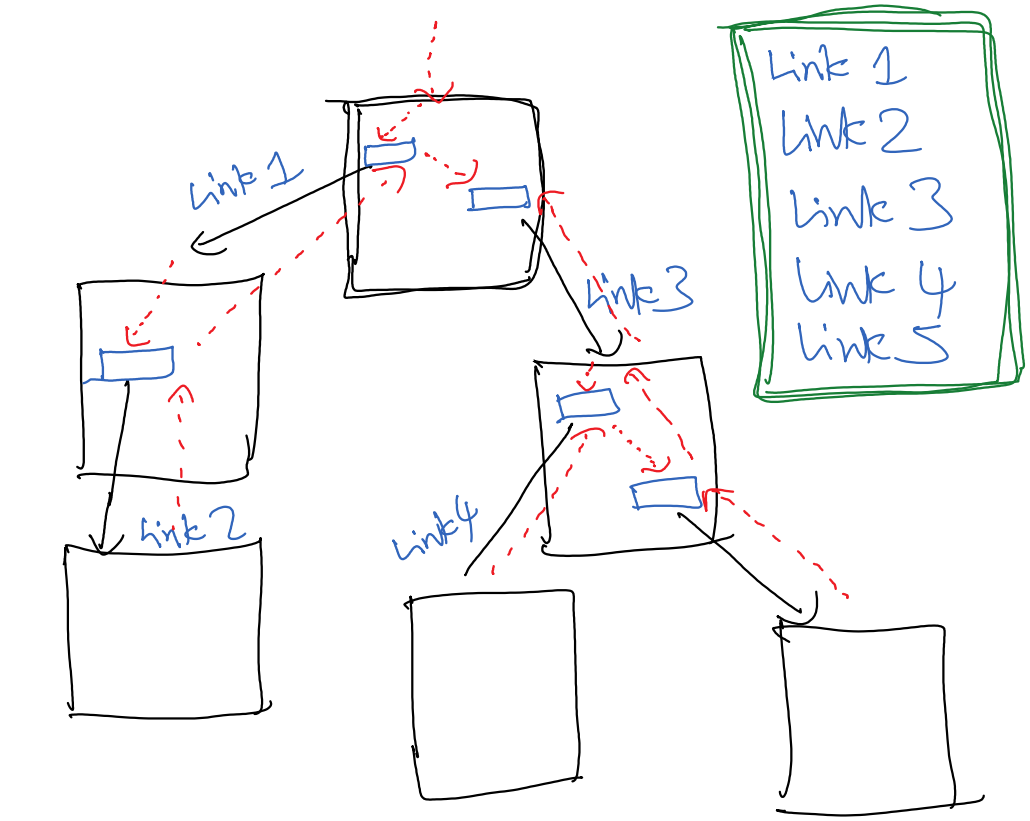
- Web Scraping targets particular data points within a webpage, such as prices, reviews, product listings, or other structured information.
- Scraping is focused on extracting certain elements or fields from a webpage, rather than exploring links or indexing the entire page.

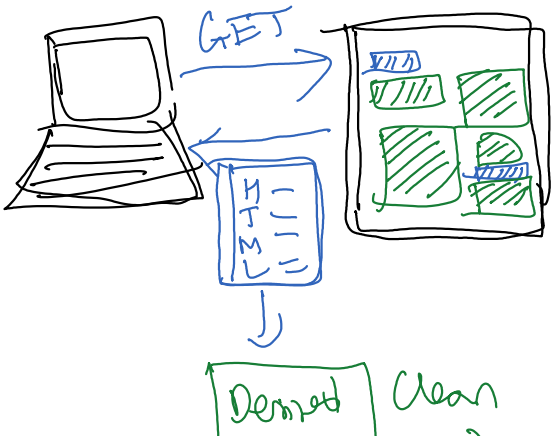


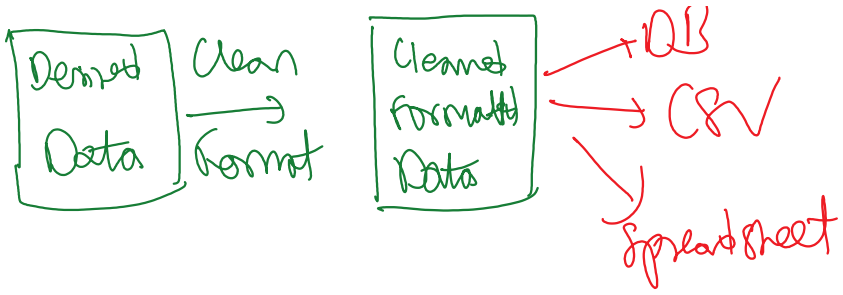
How does Web Scraping work?

1. HTTP Requests:
    - Web scrapers initiate HTTPS requests to servers to retrieve the HTML source of a webpage.
    - The GET and POST are most commonly used request types.
2. Parsing HTML:
    - The script navigates through the received HTML structure to identify and extract data of interest.
    - This involves extracting only the required specific data.
3. Storage:
    - After extraction, data is cleaned and stored in the desired format.
    - Data is usually stored in a database, CSV file, or spreadsheet for further analysis.


SIMPLE
STEPS FOR
WEB
SCRAPING
Select The Website
Find the data
Write the Script
Run the Bot
Store the Data


Link 1
Link 2
Link 3
Link 4
Link 5
Link 1
Link 3
Link 2
Link 4


GET
HTML
Denied
Clean
Cleaned
DB


Dirty
Data
Clean
Format
Cleaned
Formatted
Data
DB
CSV
Spreadsheet

Types of Web Scraping

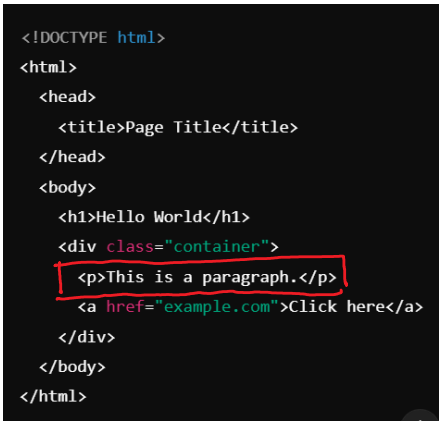
Tuesday, October 29, 2024 11:16 PM

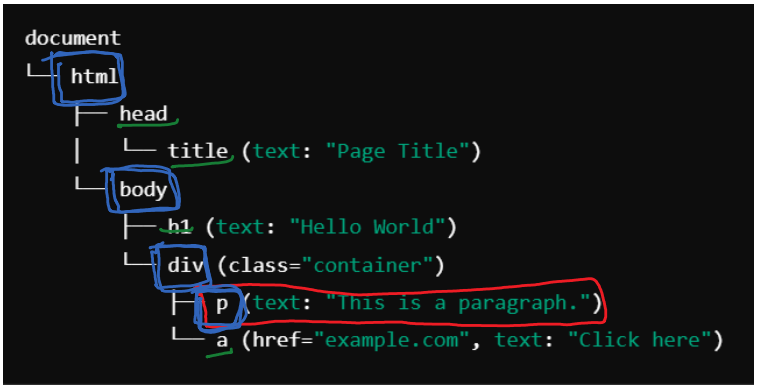
1. HTML Parsing:

- HTML parsing is the most common form of web scraping.
- It involves analyzing a web page's HTML structure to extract relevant data.
- Works well for websites with static content or basic HTML structures.
- Example: Extracting blog titles, author names, and publication dates from a blog page.

2. Data Object Model (DOM) Parsing:

- Focuses on navigating the DOM structure of a website.
- The DOM structure refers to the hierarchy of elements of the webpage.
- Works best with complex or dynamic websites where content might change upon certain events, such as clicking or scrolling.





3. Headless Browser Scraping:

- Headless browser scraping involves using a browser in headless mode to render web pages like a real user.
- There is no GUI involved in headless browsing. Nothing is display visually on the screen.
- Works best for websites that rely heavily on JavaScript or AJAX to load content.
- Puppeteer is a commonly used tool to work with headless browsers.
- Example: Extracting real-time stock prices from a financial website.

4. API-based Scraping:

- Many websites offer APIs (Application Programming Interfaces) for structured data access.
- This can be a more efficient and ethical alternative to traditional scraping methods.
- Example: Extracting user information, posts, and comments from a social media platform's API.

5. Image and Multimedia Scraping:

- Image scraping involves extracting images, videos, or other media files from web pages.
- Scrapers target img tags or other media tags in HTML, and download the files directly.

Ethical Consideration

Tuesday, October 29, 2024 11:43 PM

- Ethical considerations in web scraping are essential to ensure that data collection practices are conducted responsibly and in line with the legal and moral obligations.
- These considerations mainly revolve around respecting website policies, data privacy, intellectual property, and transparency with users.

1. **Compliance with website Terms & Services:**
    - Most websites have Terms of Service (ToS) that outline acceptable behaviors, including whether web scraping is permitted.
    - Violating these terms can result in legal repercussions, as scraping without permission may be viewed as unauthorized access.
    - It's crucial to review and abide by the website's policies and request explicit permission for data access if the site prohibits scraping.
    - **What To Do:** Before starting any scraping activity, read the website's ToS and Privacy Policy carefully. When in doubt, seek permission or use alternative, sanctioned APIs.

2. **Respect for Data Ownership and Intellectual Property Rights:**
    - The data on a website is generally owned by the website's creators or operators.
    - Unauthorized replication or distribution may infringe on intellectual property rights.
    - **What To Do:** Use scraped data strictly for purposes that do not violate intellectual property laws and avoid redistributing content without permission.

3. **Data Privacy and User Consent:**
    - Websites may contain sensitive or personal information about users, such as names, email addresses, or comments.
    - Scraping such data without explicit user consent is a privacy breach.
    - Regulations like the GDPR (Europe) and CCPA (USA) impose strict guidelines on handling personal data.
    - **What To Do:** Avoid scraping personal data unless you have explicit permission. If personal data is required, ensure compliance with relevant privacy laws.

4. **Rate Limits and Server Overload:**
    - Websites operate with limited server resources, and excessive scraping can strain servers, which can slow down performance for other users.
    - Ethical scrapers should honor the website's robots.txt file, which often specifies crawling frequency and areas off-limits to automated access.
    - **What To Do:** Implement rate limiting and time intervals between requests to reduce the impact on the website's server.

5. **Transparency and Disclosure:**
    - Ethical web scraping involves transparency about the intent and use of the data, especially if it's for commercial purposes.
    - Using data without context or presenting scraped data as a comprehensive view of a company's offerings can mislead

users and harm the reputation of the data's original source.

- **What To Do:** If using scraped data for public purposes, clearly disclose its source, the data collection process, and any limitations.

Advantages of Web Scraping

Wednesday, October 30, 2024 12:01 AM

1. **Efficient Data Collection and Processing:**
   - Web scraping allows for the automated collection of data at a large scale, offering much higher speed and efficiency than manual collection.
   - Helps save considerable time and effort, enabling faster access to information.
   - This is particularly beneficial for industries that rely on large datasets, such as e-commerce, market research, and finance.

2. **Real-Time Data Access:**
   - Web scraping enables real-time data extraction, allowing companies to monitor data and respond to changes immediately.
   - Access to real-time data provides businesses with a competitive edge by allowing them to adjust strategies based on the latest trends.

3. **Cost-Effective Market Research:**
   - Compared to traditional data collection methods, such as surveys or purchasing datasets, web scraping offers a cost-effective way to collect market data.
   - Web scraping can gather data from various websites, blogs, social media, and online forums, providing a broader view of the market landscape.

4. **Enhanced Decision-Making through Data-Driven Insights:**
   - Access to data-driven insights enables organizations to make better, evidence-based decisions.
   - Web scraping helps compile data that is crucial for understanding consumer behavior, trends, and competitor activities.
   - Helps companies analyze historical data to identify trends and predict future behaviors, aiding long-term strategy planning.

5. **Detecting and Analyzing Fraudulent Activities:**
   - By monitoring patterns in online data, web scraping can help identify potentially fraudulent activities, such as fake reviews, counterfeit product listings, or misleading advertisements.
   - Companies can use web scraping to validate information about their own products and services by comparing data across different platforms, detecting inconsistencies that may indicate fraud.

6. Enhanced SEO and Content Strategy:

- Web scraping can help companies analyze competitors' keywords, backlinks, and content strategies to improve their own SEO performance.
- Understanding high-performing content on competitors' websites can guide and allow companies to identify and replicate successful topics and formats.

Disadvantages of Web Scraping

Wednesday, October 30, 2024 12:12 AM

1. Legal and Ethical Risks:

- Many websites have terms of service that prohibit or limit data scraping.
- Extracting data without permission can lead to copyright issues, potential lawsuits, or restrictions from the website owner.
- Scraping personal information, even if publicly available, can raise privacy issues, especially under data protection laws like GDPR.
- Companies can face penalties for scraping personal data without consent.

2. IP Blocking and Bot Detection:

- Websites often deploy mechanisms like CAPTCHAs, rate limits, and IP blocking to detect and block scraping bots.
- This can interrupt scraping processes, requiring continual adjustment to circumvent these systems.
- Many scrapers use rotating proxies to avoid detection, which can be costly.
- IPs can also quickly become blocked, rendering scraping scripts useless.

3. Data Accuracy and Consistency Issues:

- Websites frequently update their layouts, URLs, or data structures.
- These changes require scrapers to be reconfigured frequently, increasing maintenance time and cost.
- Extracted data may contain inconsistencies, missing values, or irrelevant information that requires significant preprocessing before it becomes usable.
- Cleaning and standardizing such data can be time-intensive.
- Might require constant scraping and data refresh cycles

4. Incompatibility with Dynamic and JavaScript-Heavy Content:

- Many modern websites use JavaScript frameworks (like React or Angular) that load content dynamically.

- Scraping such content requires additional tools like Selenium or Puppeteer, which increase complexity.
- JavaScript-heavy pages can be slower to load and scrape, making data extraction more time-consuming and resource-demanding.

5. Environmental Impact:

- Large-scale scraping operations consume substantial computational resources, which contributes to energy usage and, indirectly, environmental impact.
- This inadvertently translates to carbon emissions, an increasingly important consideration for environmentally conscious organizations.

Alternatives to Web Scraping

Wednesday, October 30, 2024 12:13 AM

1. **Public APIs:**
   - Many websites offer public APIs that allow developers to access structured data directly.
   - APIs provide clean and organized data formats, eliminating the need for extensive parsing or cleaning.
   - Using an official API helps avoid legal risks associated with web scraping.

2. **RSS Feeds:**
   - Really Simple Syndication feeds are a way to automatically receive updates from websites in a single feed.
   - RSS feeds are updated frequently, making it easy to access new content automatically.
   - Since RSS feeds are structured in XML, they're easy to parse and don't require complex scraping scripts.

3. **Public Datasets:**
   - Data portals provide clean, verified, and well-documented datasets, which are typically updated periodically.
   - Most data portals offer free access, with datasets available in formats like CSV, JSON, or Excel.
   - Using existing datasets reduces time spent on collection and cleaning.

4. **Manual Data Collection:**
   - No technical setup or coding is needed, making it accessible to anyone who can access the site.
   - Can be efficient without the need for dedicated tools or servers.
   - It often avoids triggering anti-scraping measures.

5. **Licensed Partnerships with Data Owners:**
   - Partnerships can unlock data that is not available publicly, providing a competitive edge.
   - Data is usually provided in structured formats and with reliable update frequencies, making it easy to integrate.
   - Since data is obtained through agreements, this avoids any compliance issues.

Topics

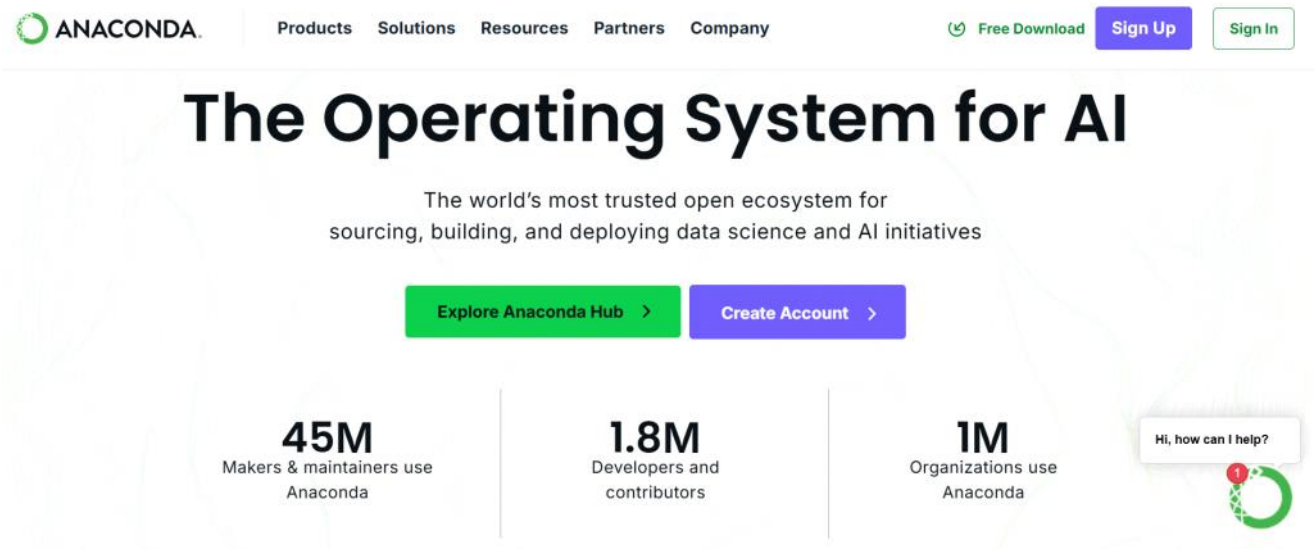
Sunday, November 3, 2024 12:44 PM

1. **About Anaconda**
2. **Tutorial of Common Anaconda Prompts**
3. **Creating a project environment using Anaconda**

About Anaconda

Thursday, October 31, 2024 10:32 AM

- Anaconda is a popular open-source distribution of Python (and R) mainly used for data science, machine learning, and scientific computing.
- Anaconda includes Conda, a package, dependency, and environment manager, making it easy to install and manage libraries and environments.
- It comes with over 1,500 pre-installed packages for data science, including popular libraries like NumPy, Pandas, Matplotlib, TensorFlow, and Scikit-Learn.
- Anaconda provides easy access to Jupyter Notebook, a powerful tool for interactive coding, data visualization, and exploratory analysis.
- Anaconda includes conda, which is its package and environment manager (similar to pip)



Common Anaconda Prompts

Thursday, October 31, 2024 10:46 AM

Common Anaconda Prompts:

- **conda update conda**: updates conda to the latest version
- **conda update -all**: updates all packages to the latest version
- **conda env list**: lists all available environments
- **conda create --name <env_name>**: create a new environment
- **conda activate <env_name>**: activates an environment
- **conda deactivate**: deactivates the current working environment
- **conda list**: lists all the packages installed in the current environment
- **conda env export --name <env_name> --file environment.yml:** export an environment to a .yml file
- **conda env create --file environment.yml:** import the exported environment in another system
- **conda remove --name <env_name> --all**: removes the provided environment

Setup

Thursday, October 31, 2024 9:39 AM

1. Create a project directory

2. Install Anaconda
   - Official Website

3. Open Anaconda Prompt

4. Create an Anaconda environment

5. Activate the created environment

6. Install necessary packages:
   - **pandas**
   - **numpy**
   - **requests**
   - **beautifulsoup4**
   - **lxml**: recommended for parsing XML/HTML content
   - **html5lib**: alternate parsers for Beautiful Soup
   - **selenium**
   - **python-chromedriver-binary**: Chrome driver for Selenium
   - **python-geckodriver**: Firefox driver for Selenium
   - **webdriver-manager**: manages and downloads web drivers automatically (*recommended, for auto-updates*)
   - **jupyter**: installs the main components of the Jupyter ecosystem
   - **ipykernel**: to create a jupyter kernel for an environment

6. Create an appropriate Jupyter kernel:
   - **python -m ipykernel install --user --name=<env_name> --display-name "<Your Env Display Name>"**

7. Launch Jupyter and create new notebooks using the appropriate kernel

Topics

Thursday, October 31, 2024 12:07 PM

1. **Client-Server model**
2. **HTTP Request and Response**
3. **HTTP Methods**
4. **Status Codes**
5. **Web Technologies**
    - *HTML*
    - *CSS*
    - *JavaScript*

Client-Server model/architecture

Thursday, October 31, 2024 12:14 PM

- The Client-Server model is a fundamental design framework for networked applications.
- It organizes interactions between two major entities: clients (requesters) and servers (providers of resources).
- This architecture underpins most modern networks, including the internet, web applications, email systems, and various enterprise systems.

Client:

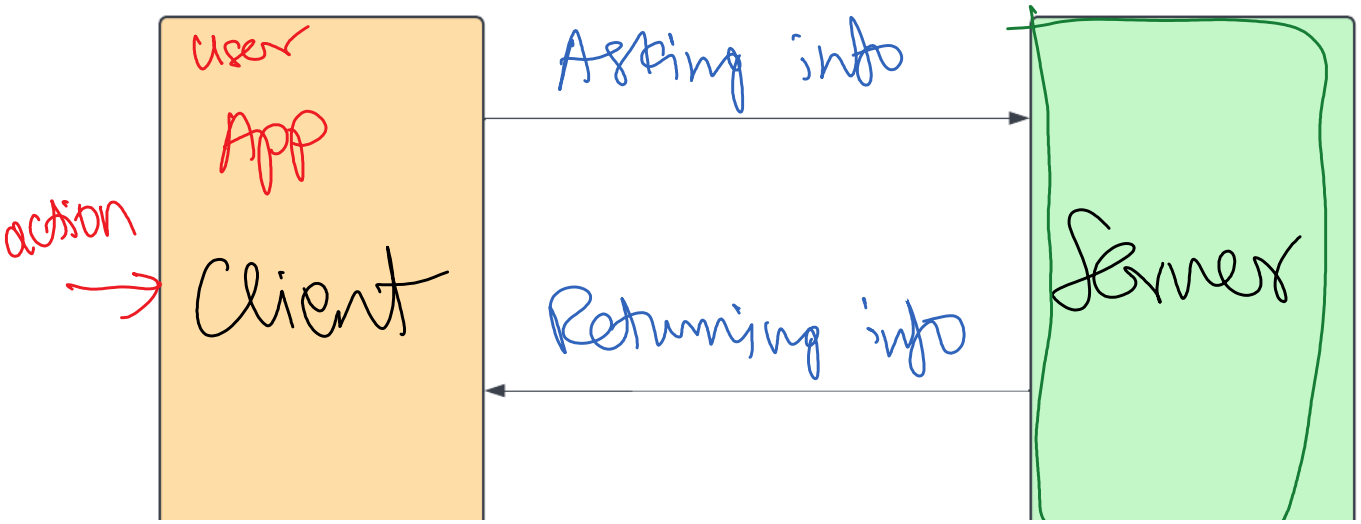
- Device or an application that initiates requests for services or resources.
- Clients are typically end-user devices (e.g., smartphones, laptops) or software applications (e.g., browsers, email clients) that communicate over a network.

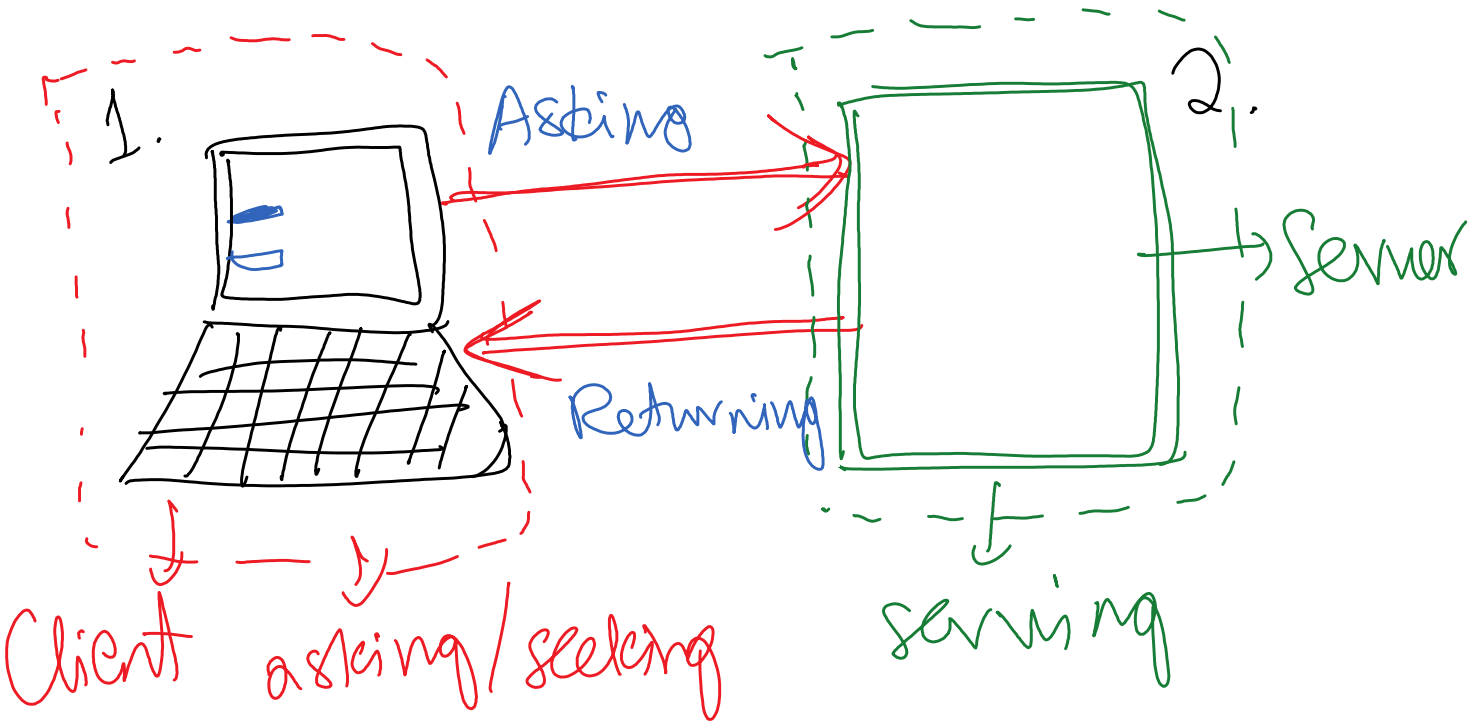
Server:

- A server is a dedicated system or application that listens for and fulfills requests from clients.
- Servers provide resources, data, or services to clients, typically through a network connection.

Working:

- *importance of server for websites/apps*
- *client-server communication*




1.
Asking
2.
Server
Returning
Client
asking/seeking
serving

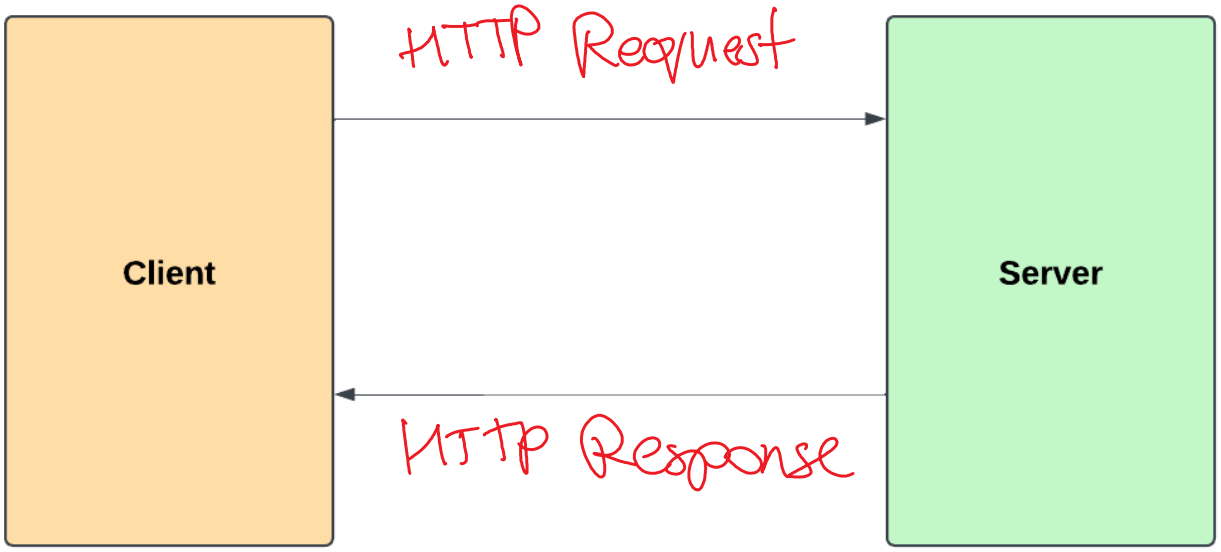
HTTP Request and Response

Thursday, October 31, 2024 12:14 PM

HTTP:

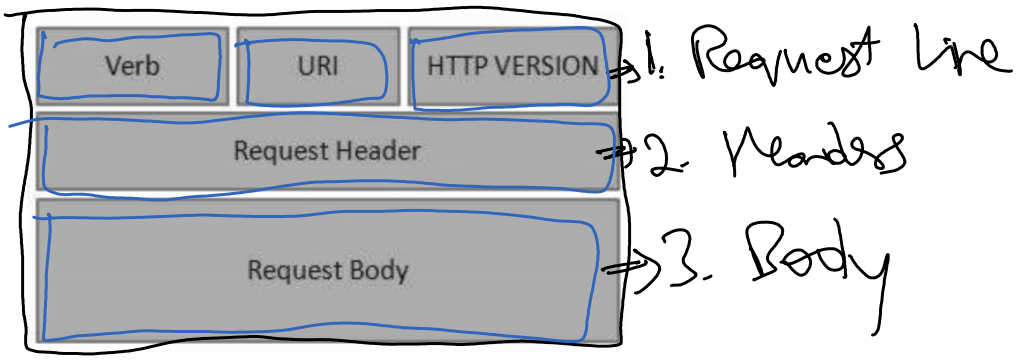
- The HTTP (Hypertext Transfer Protocol) is the foundation of data communication on the web.
- It facilitates the exchange of information between clients and servers.
- The HTTP request-response cycle is central to how web applications function.

Working:



HTTP Request:

- An HTTP request is a message sent by the client to the server to initiate an action or request a resource.
- The request message consists of following components:
  - **Request Line**:
    - Method / Verb
    - Address (URI)
    - Version
  - **Headers**: used for conveying additional meta-data about the request
  - **Body**: contains the major contents of the request message



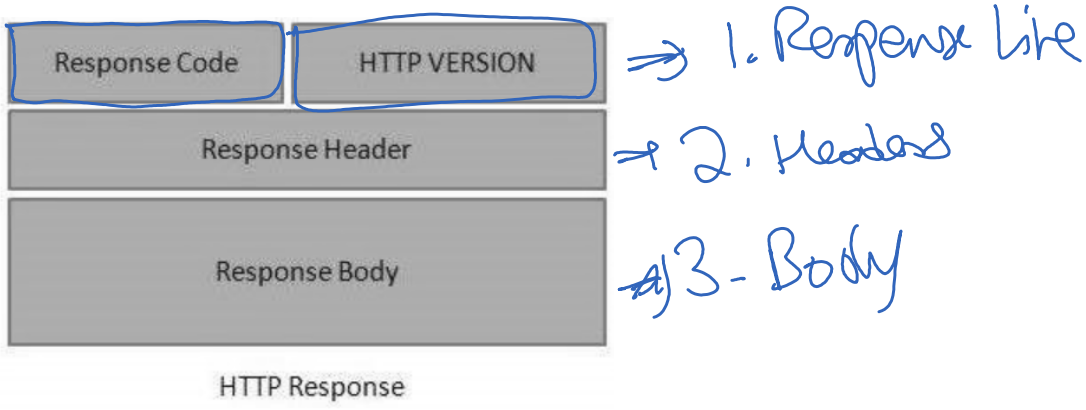
HTTP Request

- Example Request message:

```
POST /api/login HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Accept: application/json
Content-Type: application/json
Content-Length: 55

{
  "username": "john_doe",
  "password": "securepassword123"
}
```

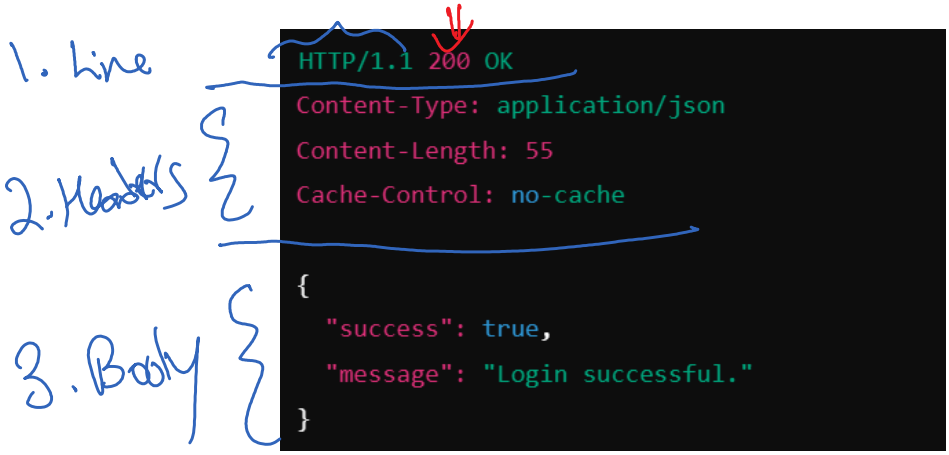
HTTP Response:

- An HTTP response is the message sent by the server back to the client after processing the request.
- The response message consists of following components:
  - **Response Line**:
    - Status Code
    - Version
  - **Headers**: used for conveying additional meta-data about the response
  - **Body**: contains the requested contents (usually JSON format)



HTTP Response

- Example Response message:



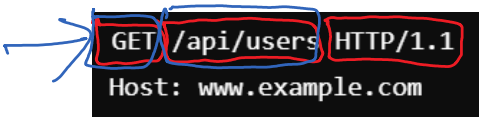
HTTP Methods

Thursday, October 31, 2024 12:14 PM

- HTTP methods, also known as HTTP verbs, are a fundamental part of the HTTP protocol.
- They define the action to be performed on a resource identified by a URI.
- Each method has specific semantics and is used for different purposes in client-server communication.

1. GET:

- Used to request data from a specified resource.
- It is the most commonly used HTTP method.
- **Application**: Retrieving web pages, images, or other resources from a server.



2. POST:

- Used to submit data to be processed to a specified resource.
- This often results in the creation of a new resource.
- **Application**: Submitting forms or uploading files

3. PUT:

- Used to update an existing resource or create a new resource if it does not exist.
- It sends data to the server to replace the current representation of the resource.
- **Application**: Updating user details or replacing an entire user.

```
Content-Type: application/json

{
  "username": "john_doe",
  "email": "john_doe@example.com"
}
```

4. PATCH:

- Used to apply partial modifications to a resource.
- It sends a set of instructions to update the resource rather than replacing it entirely.
- **Application**: Updating specific fields of a user, like changing a user's email without altering other attributes.

```
PATCH /api/users/1 HTTP/1.1
Host: www.example.com
Content-Type: application/json

{
  "email": "john_updated@example.com"
}
```

5. DELETE:

- Used to remove a specified resource from the server.
- **Application**: Deleting user accounts, posts, or other resources.

```
DELETE /api/users/1 HTTP/1.1
Host: www.example.com
```

Summary of HTTP Methods:

METHOD	PURPOSE	USE CASES
GET	Retrieve data	Fetching pages or resources
POST	Submit data to create/update	Submitting forms, creating data
PUT	Update/replace resource	Updating existing resources
PATCH	Partially update resource	Modifying specific fields
DELETE	Remove a resource	Deleting resources
HEAD	Retrieve headers only	Checking resource metadata
OPTIONS	Describe communication options	Discovering server capabilities
TRACE	Diagnostic, echo the received request	Debugging requests

HTTP Status Codes

Thursday, October 31, 2024 12:14 PM

- HTTP status codes are three-digit numbers sent by a server in response to a client's request made to the server.
- These codes are crucial for understanding the results of HTTP requests.
- They provide important feedback to clients about the success or failure of requests and help developers diagnose issues.
- Properly using and interpreting HTTP status codes is essential for effective communication between clients and servers in the web ecosystem.

1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

1. 1xx - Informational:

- These codes indicate that the request has been received and the process is continuing.
- They are rarely used in web applications but are important for certain protocols.
- **Examples**:
  - *100 Continue*
  - *101 Switching Protocols*

2. 2xx - Success:

- These codes indicate that the client's request was successfully received, understood, and accepted.
- **Examples**:
  - *200 OK*
  - *201 Created*
  - *202 Accepted*

3. 3xx - Redirection:

- These codes indicate that further action is needed to complete the request.

- This usually involves redirection to another URI.
- **Examples**:
  - *300 Multiple Choices*
  - *301 Moved Permanently*
  - *302 Found*

4. 4xx - Client Error:

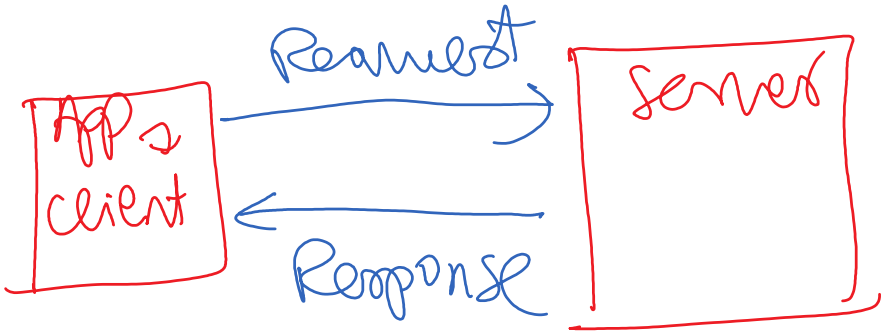
- These codes indicate that the client made an error, resulting in the request being unable to be fulfilled.
- **Examples**:
  - *400 Bad Request*
  - *401 Unauthorized*
  - *404 Not Found*

5. 5xx - Server Error:

- These codes indicate that the server failed to fulfill a valid request due to an error on the server side.
- **Examples**:
  - *502 Bad Gateway*
  - *503 Service Unavailable*

Summary of HTTP Status Codes:

Status Code	Description	Category
100	Continue	Informational
200	OK	Success
201	Created	Success
204	No Content	Success
300	Multiple Choices	Redirection
301	Moved Permanently	Redirection
302	Found	Redirection
400	Bad Request	Client Error
401	Unauthorized	Client Error
403	Forbidden	Client Error
404	Not Found	Client Error
500	Internal Server Error	Server Error
503	Service Unavailable	Server Error


Request
App's client
server
Response

Web Technologies

Thursday, October 31, 2024 12:15 PM

- Web technologies encompass a wide array of tools, languages, and protocols used to create, maintain, and manage websites and web applications.
- HTML provides the foundational structure, CSS handles visual presentation, and JavaScript adds dynamic behavior.
- Together, they enable developers to build rich, interactive user experiences on the web.
- Understanding these technologies is crucial for web scraping, as it allows developers to extract data from web pages effectively, even when dealing with dynamic content.

1. HTML:

- HTML is the standard markup language used for creating web pages.
- It provides the structure and layout of a web document by defining elements like headings, paragraphs, links, images, and other types of content using appropriate tags.
- Tags can also be nested to create more complex structures.

```html
<html>
<head>
    <title>My Web Page</title>
</head>
<body>
    <header>
        <h1>Welcome to My Website</h1>
    </header>
    <section>
        <p>This is a paragraph of text on my website.</p>
        <a href="https://example.com">Visit Example.com</a>
    </section>
    <footer>
        <p>&copy; 2024 My Website</p>
    </footer>
</body>
</html>
```

2. CSS:

- CSS is a stylesheet language used to describe the presentation of a document written in HTML.
- It controls the layout, colors, fonts, and overall visual appearance of web pages.

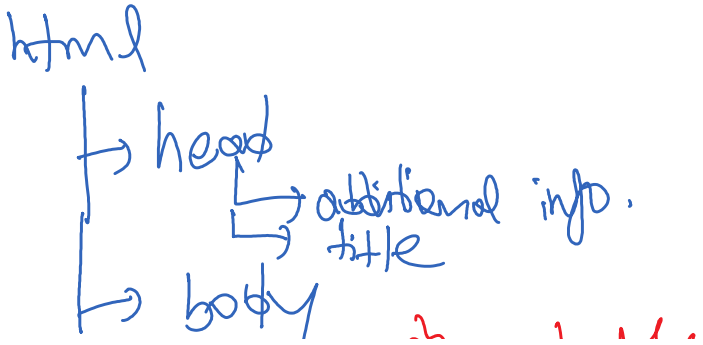
- CSS is a stylesheet language used to describe the presentation of a document written in HTML.
- It controls the layout, colors, fonts, and overall visual appearance of web pages.
- CSS uses selectors to target HTML elements and apply styles.

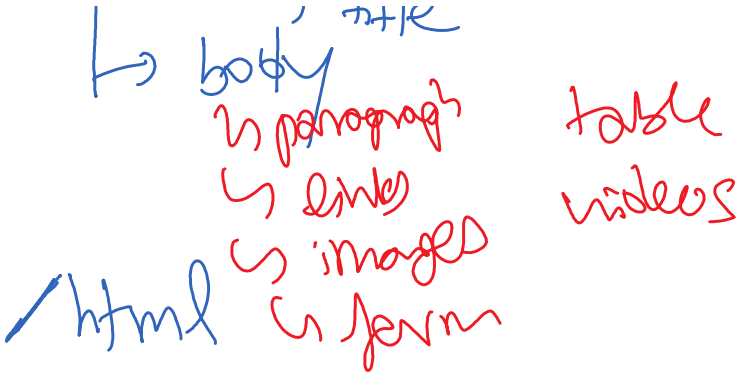
```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

header {
    background-color: #f8f9fa;
    padding: 20px;
    text-align: center;
}

h1 {
    color: #333;
}
```

3. JavaScript:

- JavaScript is a high-level, dynamic programming language that enables interactivity and functionality on web pages.
- It allows developers to create rich user experiences through client-side scripting.

```javascript
document.addEventListener("DOMContentLoaded", function() {
    const button = document.querySelector("button");
    button.addEventListener("click", function() {
        alert("Button clicked!");
    });
});
```




body
paragraph
link
images
form
table
videos
html

About Requests Module

Thursday, October 31, 2024 2:32 PM

- The Requests module is a powerful and user-friendly HTTP library for Python, designed to make it easier to send HTTP requests.
- It abstracts away the complexities of making requests and handling responses, allowing developers to focus on building applications.

Key Features:

- **Simplicity:** Known for its clean and straightforward syntax, making it accessible for beginners and experienced developers alike.
- **Flexibility:** Supports a wide range of HTTP methods and allows for customization, such as adding headers, query parameters, and more.
- **User-Friendly:** Compared to the built-in urllib library, Requests provides a more intuitive API which is particularly beneficial for rapid development and prototyping.
- **Automatic Content Decoding:** Requests automatically decodes the content based on the response headers, so you can work with the data directly without worrying about the encoding.
- **Session Management:** You can persist certain parameters across multiple requests using session objects, which can be useful for maintaining state.
- **Built-in Error Handling:** Requests come with built-in mechanisms to handle common HTTP errors and exceptions.
- **Community & Support:** The Requests library has a large and active community. Can easily find tutorials, documentation, and support for any issues you encounter.

Beloved Features

Requests is ready for today's web.

- Keep-Alive & Connection Pooling
- International Domains and URLs
- Sessions with Cookie Persistence
- Browser-style SSL Verification
- Automatic Content Decoding
- Basic/Digest Authentication
- Elegant Key/Value Cookies
- Automatic Decompression
- Unicode Response Bodies
- HTTP(S) Proxy Support
- Multipart File Uploads
- Streaming Downloads
- Connection Timeouts
- Chunked Requests
- `.netrc` Support

Requests officially supports Python 3.8+, and runs great on PyPy.

Applications of Requests:

1. Web Scraping:

- Requests is often the first step in web scraping.
- It allows developers to send HTTP requests to retrieve the HTML content of web pages, which can then be parsed and analyzed using libraries like Beautiful Soup or lxml.

2. API Interaction:

- Commonly used to interact with RESTful APIs.
- It can send various types of HTTP requests (GET, POST, PUT, DELETE) to perform operations like retrieving data, creating new records, or updating existing ones.

3. File Uploads:

- Requests supports file uploads, which is essential for applications where users need to submit files to a server.

4. Session Management:

- Requests provides the ability to maintain sessions across multiple requests, which is useful for applications that require authentication.
- A web application that requires users to log in can use a session to maintain the user's login state while making subsequent requests.

5. Data Submission:

- Requests can be used to submit data to web servers, especially in web forms where users enter information.

- Example: An application could use Requests to send user feedback or comments to a server.

GET requests

Thursday, October 31, 2024 5:47 PM

Basic Syntax:

```
import requests

response = requests.get("<uri>")
```

Query Parameters:

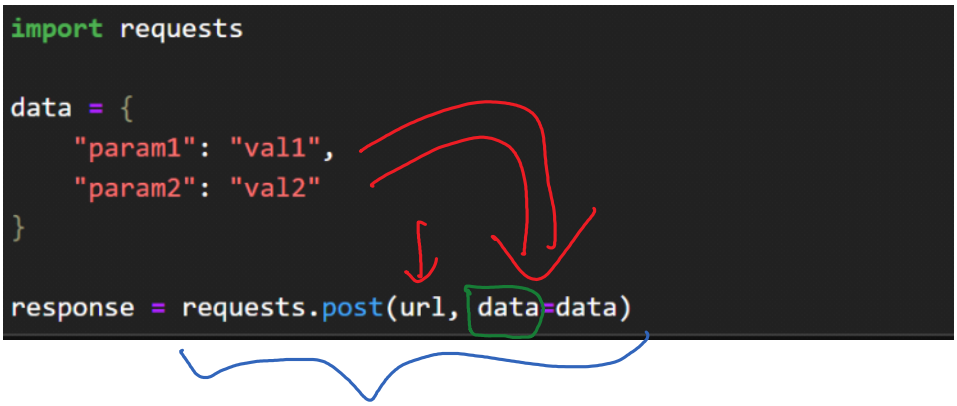
- They are parameters which are included in the URI as part of the request message.
- Mainly used to filter the data received from the server.
- In a URI, these are placed after the ? and separated by &
- **Application**: Filter phones by a price range in an e-commerce website

www.flipkart.com?product=phones&min_price=10000&max_price=30000

POST requests

Thursday, October 31, 2024 6:28 PM

Basic Syntax:



- Since POST requests are used to send data to the server to create a resource, the Requests library provides the data parameter for this
- We can also pass JSON data directly using the json parameter
- These parameters accept dictionary-like objects as arguments

Headers

Thursday, October 31, 2024 7:33 PM

- Headers enable communicate of additional information such as authorization credentials, content types, and user-agent data, etc. between the clients and servers.
- Understanding and managing headers is essential for authenticating requests, specifying response formats, handling cookies, and personalizing user experiences.

Common types of Headers:

- **Authorization**: Used for passing authentication tokens.
- **Content-Type**: Specifies the format of the request data, such as JSON or XML.
- **User-Agent**: Provides information about the client making the request, such as the browser or app.
- **Accept**: Informs the server about the types of content the client can process, like JSON or HTML.
- **Custom Headers**: Headers that may be unique to an API or web application.

Practical tips for using Headers:

- Always check API documentation to see required headers for a particular request.
- Use headers for efficient server interaction because servers expect headers for security, data formatting, and client identification.
- Perform appropriate error handling because missing or incorrect headers can lead to HTTP errors like 401 Unauthorized or 415 Unsupported Media Type.

About Beautiful Soup

Saturday, November 2, 2024 9:39 AM

- Beautiful Soup is a Python library used to parse HTML and XML documents.
- It's especially useful for web scraping because it helps navigate, search, and modify the HTML (or XML) content fetched from a webpage.
- It transforms complex HTML documents into a tree structure, where each node corresponds to elements such as tags, text, attributes, etc.
- This makes it easy to locate and extract specific information.

Advantages of Beautiful Soup:

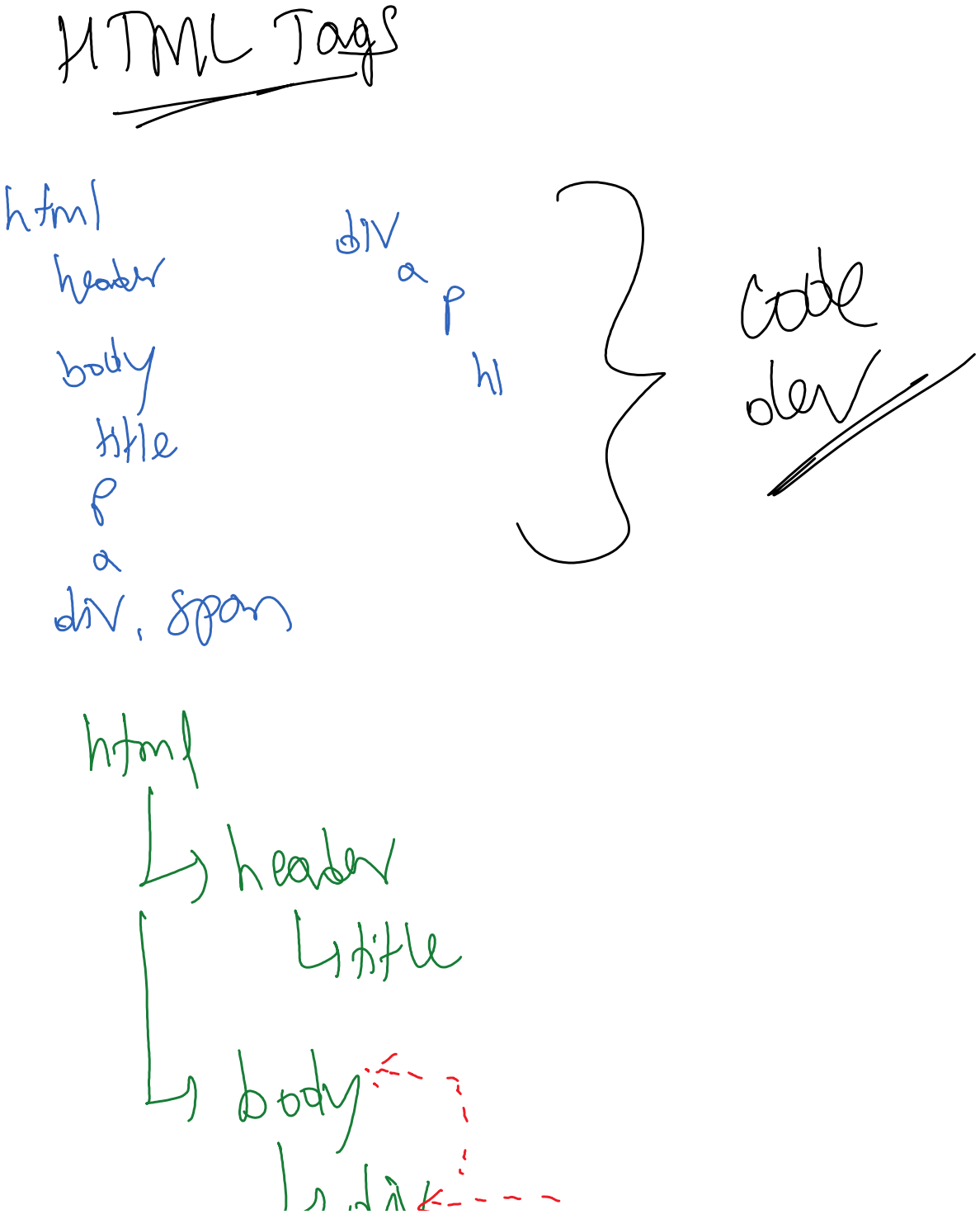
- **Easy to Learn and Use**: Has a user-friendly syntax that makes it easy for users to quickly locate and extract data from web pages.
- **Flexible Parsing**: Works with different parsers, such as the built-in Python parser or lxml, offering flexibility in terms of speed and error handling.
- **Handles Broken HTML**: Automatically fixes errors in the HTML structure, allowing users to scrape data from pages that other parsers might struggle with.
- **Efficient Navigation and Search Functions**: Provides intuitive functions like find, find_all, and select to search and navigate through HTML tags and CSS selectors.
- **Integration with Other Libraries**: Integrates smoothly with libraries like Requests, to retrieve web pages before parsing them. Also works well with Pandas for data analysis or Selenium for JavaScript-heavy pages, making it a versatile choice for a complete web scraping workflow.
- **Well-Documented and Active Community**: Has a comprehensive documentation and an active community that provides resources, tutorials, and troubleshooting support, making it accessible for new users.

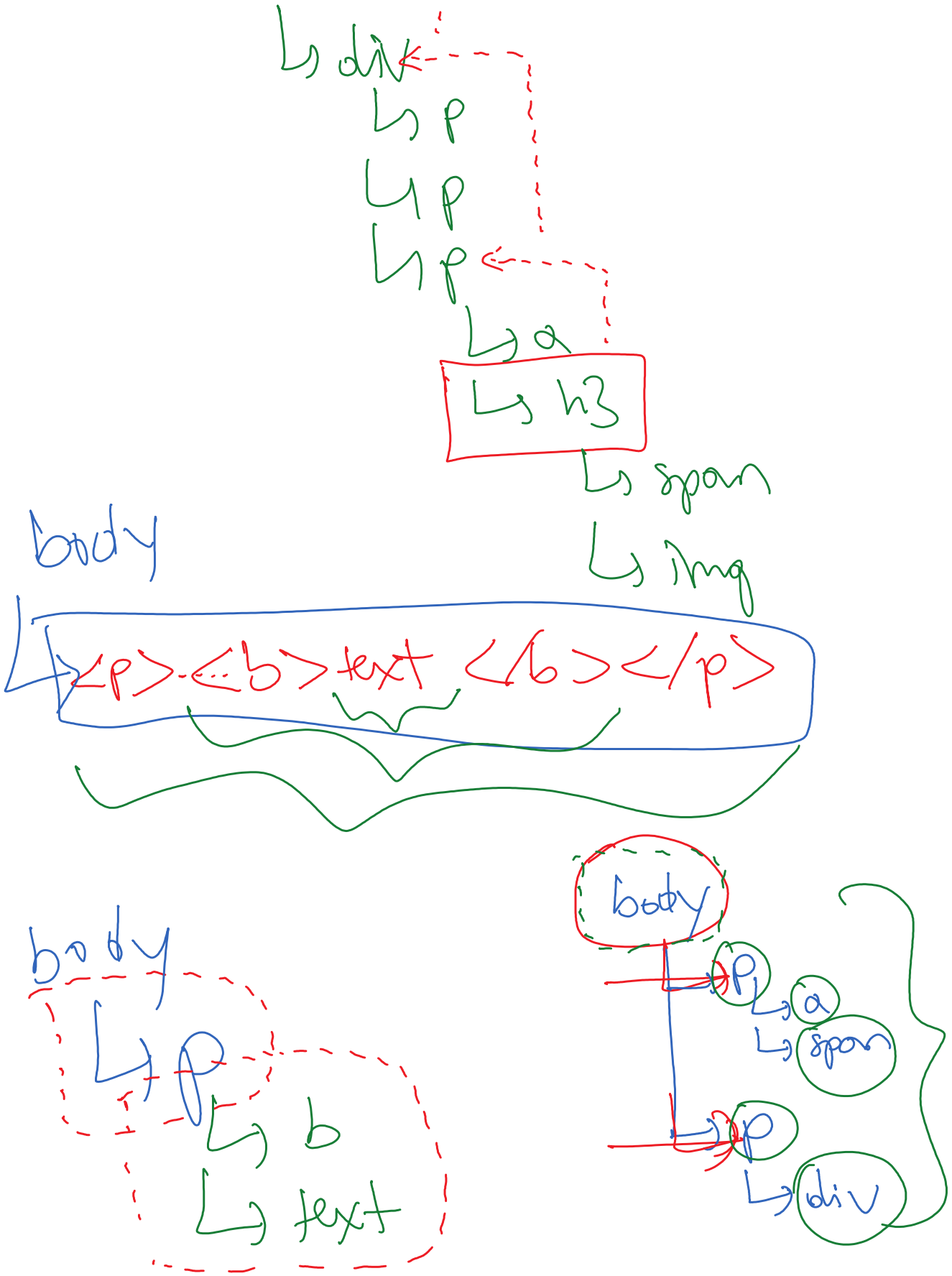
Applications of Beautiful Soup:

- **Price Comparison and Monitoring**: Widely used by e-commerce companies and consumers to scrape prices from various online stores.
- **Job Listings Aggregation**: Commonly used to scrape job listings from platforms like LinkedIn, Indeed, or company career pages. This can help create job aggregators that compile positions from various sources.
- **Market Research and Sentiment Analysis**: Companies often use web scraping to collect data from forums, blogs, and review sites to analyze customer sentiment about their products or their

competitors.

- **Real Estate Listings**: Useful for gathering real estate listings from sites like Zillow or Realtor.com. Data on prices, locations, features, and property availability can be scraped and analyzed to identify trends, track prices, and help potential buyers and real estate investors make informed decisions.
- **Travel and Flight Price Tracking**: Used to monitor and compare prices across different airlines, hotels, and booking platforms. By gathering this data, users can develop apps to track flight and accommodation prices, helping travelers find the best deals.




div
p
p
p
a
h3
span
img
body
text
body
p
b
text
body
p
a
span
p
div

Topics

Tuesday, November 5, 2024 11:14 PM

1. **About Selenium**

2. **Getting Started**

3. **Navigating Web Pages**
    - *Locating Web Elements*
    - *Understanding Xpath*
    - *Interacting with Web Elements*
    - *Dropdown & Multiselect*
    - *Scrolling*

4. **Advanced Web Interaction**
    - *Explicit Waits*
    - *Implicit Waits*
    - *Handling Alerts*
    - *Frames & iFrames*

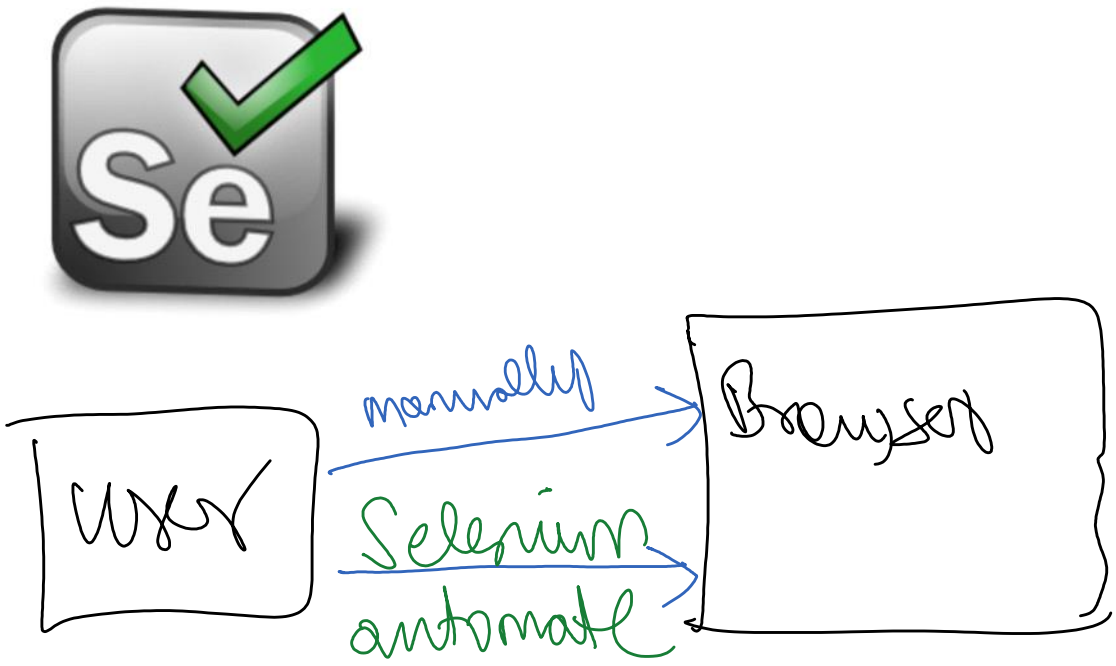
5. **Best Practices & Optimization**

1. Definition

Tuesday, November 5, 2024 11:14 PM

What is Selenium?

- Selenium is a powerful, open-source tool used for automating web browsers.
- It is often utilized for web scraping when interacting with dynamic websites that rely on JavaScript to load content, which static scraping libraries like Beautiful Soup or Requests cannot handle effectively.
- When scraping websites using Python, Selenium acts as a web driver, automating browser actions to interact with web pages like a human user.
- It can navigate to web pages, simulate user interactions (clicks, scrolls, form submissions), and extract data directly from rendered HTML.
- Selenium was originally developed for testing web applications. Over time, it became a popular tool for web scraping due to its ability to handle dynamic, JavaScript-heavy websites.

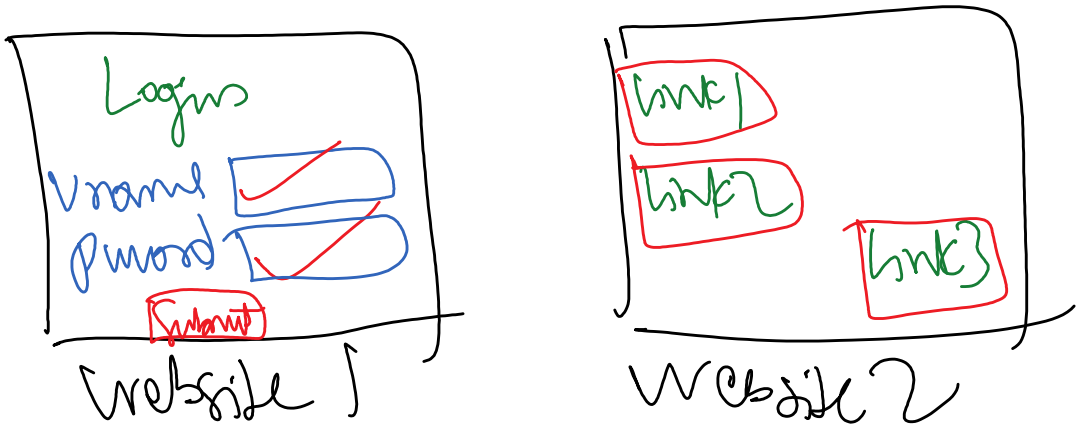


2. Key Features

Monday, December 2, 2024 11:41 PM

Key Features:

- **Dynamic Content Handling:**
  - Can scrape data from JavaScript-heavy sites.
  - Waits for elements to load before interacting or extracting data.
- **Interaction Simulation:**
  - Handles tasks such as clicking buttons, filling forms, selecting dropdowns, and scrolling pages.
  - Useful for scraping data hidden behind user interactions.
- **Cross-Browser Support:**
  - Works with popular browsers like Chrome, Firefox, Edge, and Safari.
- **Customizable Waits:**
  - Implements explicit and implicit waits to ensure elements are fully loaded before actions are performed.



3. Comparison

Monday, December 2, 2024 11:41 PM

Comparative Analysis:

FEATURE	SELENIUM	BEAUTIFUL SOUP	REQUESTS
Handles JavaScript	✅	❌	❌
Ease of Use	Moderate	Easy	Easier
Speed	Slower (browser overhead)	Faster	Fastest
Interaction with Elements	✅	❌	❌

4. Advantages

Monday, December 2, 2024 11:42 PM

Advantages:

- Handles JavaScript and AJAX (*Asynchronous JavaScript and XML*)
- Simulates user behavior
- Allows scraping data behind login screens or requiring user interaction

5. Disadvantages

Monday, December 2, 2024 11:42 PM

Disadvantages:

- Slower than static scraping methods since it requires a full browser environment
- Heavy CPU and memory usage
- Websites may block or detect Selenium bots

1. First Steps

Monday, December 2, 2024 11:47 PM

1. Activate virtual environment
2. Install selenium
3. Download suitable web driver

2. Exercise

Tuesday, December 3, 2024 12:01 AM

Write a Python script to:

- Initialize a web driver
- Open Google's home page
- Get the below details:
    - *Title of webpage*
    - *URL of webpage*
- Take a screenshot of the page
- Close the browser

1. Locating Elements

Tuesday, December 3, 2024 12:35 AM

- Selenium provides the function find_element to find and locate elements of a webpage
- There're several strategies to identify an element of a webpage:

1. **ID:**

```
element = driver.find_element("id", "<element_id>")
```

2. **Name:**

```
element = driver.find_element("name", "<element_name>")
```

3. **Class:**

```
element = driver.find_element("class name", "<element class id>")
```

4. **Tag:**

```
element = driver.find_element("tag name", "<element tag>")
```

5. **XPath:**

```
element = driver.find_element("xpath", "<element xpath link>")
```

Element Location Strategies:

Locator Method	Locator Class	Description
id	By.ID	Finds an element by ID.
name	By.NAME	Finds an element by name attribute.
class name	By.CLASS_NAME	Finds an element by CSS class name.
tag name	By.TAG_NAME	Finds elements by HTML tag.

Locator Method	Locator Class	Description
id	By.ID	Finds an element by ID.
name	By.NAME	Finds an element by name attribute.
class name	By.CLASS_NAME	Finds an element by CSS class name.
tag name	By.TAG_NAME	Finds elements by HTML tag.
css selector	By.CSS_SELECTOR	Finds elements using CSS selectors.
link text	By.LINK_TEXT	Finds links by their exact text.
partial link text	By.PARTIAL_LINK_TEXT	Finds links by partial match of their text.
xpath	By.XPATH	Finds elements using XPath expressions.

```python
from selenium.webdriver.common.by import By

element = driver.find_element(By.ID, "<element_id>")
```

2. XPath

Tuesday, December 3, 2024 12:58 AM

What is XPath?

- XPath (XML Path Language) is a query language used to navigate and locate elements within XML or HTML documents.
- Selenium uses XPath as one of its locator strategies to find elements on a webpage.
- XPath is a powerful tool for locating elements in Selenium, offering unmatched flexibility and precision.
- It's a go-to solution when working with complex web pages or when other locators are insufficient.

Advantages:

- **Locate Elements Anywhere:**
  - XPath can traverse the entire DOM, allowing you to locate elements in deep nested structures or without unique identifiers.
  - Works for all elements, even those without id, name, or class.
- **Offers Rich Syntax:** XPath supports a variety of conditions and operators, enabling users to
  - Match elements by attributes
  - Locate elements by position
  - Use partial matches
- **Supports Text-Based Matching**:
  - Locate elements based on visible text
- **Supports Relative Paths**: You can locate elements without specifying their full path in the DOM, making XPath expressions robust to changes
  - Relative: //div[@class='example']
  - Absolute: /html/body/div[1]/div[2]
- **Navigate the DOM Hierarchy**

```
//div[@id='container']/child::p  # Direct child
//span[@class='item']/ancestor::div  # Ancestor
//div[@id='content']/following-sibling::div  # Sibling
```

- **Independent of HTML Structure**
  - XPath can navigate through the DOM and locate elements that might not be directly visible or styled.
  - Elements in the DOM can exist even if they are hidden from the user's view, such as elements with CSS properties like display: none or visibility: hidden

Disadvantages:

- XPath is slower than CSS Selectors because of its ability to traverse the entire DOM.
- XPath expressions can be harder to read and maintain, especially for deeply nested elements.
- Some older browsers may have limited support for advanced XPath queries.

3. Interacting with Elements

Tuesday, December 3, 2024 1:16 AM

1. Typing Input into Fields:

- This is achieved using the send_keys function
- It's used to simulate typing into an element, such as a text input field or a text area.
- It allows users to send keystrokes or input text programmatically as if a user were typing on a keyboard.
- Enables simulation of key presses like Enter, Tab, etc.
    - *This is achieved by using the class Keys from the module selenium.webdriver.common.keys*

2. Clearing Input Fields:

- This is achieved using the clear function
- It's used to clear the text content of a text input element on a web page
- It ensures the field is empty before entering new data (which can be done using the send_keys function)

3. Clicking Buttons:

- This is achieved using the click function
- It's used to simulate a mouse click on a web element
- Allows users to interact with clickable elements on a web page, such as buttons, links, checkboxes, or radio buttons.

4. Submitting Forms:

- This is achieved using the submit function
- Helps to automatically submit a form without explicitly clicking the "Submit" button
- Executes the action associated with the <form> tag, such as navigating to a new page or processing data.
- Direct use of submit is not common in modern web development:
    - *Most forms today rely on JavaScript events or custom logic*
    - *For such forms, using the click function on a "Submit" button or JavaScript execution may be more reliable.*

4. Dropdown & Multiselect

Tuesday, December 3, 2024 10:29 PM

1. Dropdown:

- Need to identify the dropdown element as usual
- Wrap the identified element under the class Select, imported from the module selenium.webdriver.support.select
- There are 3 ways to select an option from the dropdown list:
  - **select_by_index**: provide the index of the option
  - **select_by_value**: provide the name attribute of the option
  - **select_by_visible_text**: provide the actual text value of the option

2. Multiselect:

- Similar to dropdown as discussed above
- There are multiple ways to select an option:
  - **select_by_index**: provide the index of the option
  - **select_by_value**: provide the name attribute of the option
  - **select_by_visible_text**: provide the actual text value of the option
- Similarly, there are multiple ways to deselect an option:
  - **deselect_by_index**: provide the index of the option
  - **deselect_by_value**: provide the name attribute of the option
  - **deselect_by_visible_text**: provide the actual text value of the option
  - **deselect_all**: takes no arguments

5. Scrolling

Thursday, December 5, 2024 12:19 AM

- There're several ways of scrolling a webpage using Selenium:
  - *Scrolling to a specific element*
  - *Scrolling vertically*
  - *Scrolling horizontally*
  - *Scrolling the page height*
  - *Infinite scrolling*
- Scrolling actions are mainly achieved using the execute_script method

```
driver.execute_script(script, *args)
```

  - This method is mainly used to execute JavaScript code within the context of the currently loaded webpage
  - It allows users to directly interact with and manipulate the Document Object Model (DOM) of the page
  - Helps interact with elements that might not be accessible using Selenium's standard methods
  - To better handle dynamically loaded content on modern and JavaScript-heavy websites
    - ***script:*** String containing JavaScript code
    - ***args:*** Optional arguments to pass to the JavaScript code, usually Web Elements or other variables

1. Scrolling to a specific element:

- Identify any element on the webpage
- Use the method scrollIntoView

```
element = driver.find_element("id", "specificElementId")
driver.execute_script("arguments[0].scrollIntoView();", element)
```

2. Scrolling Vertically:

- Use the method scrollBy
- Specify the number pixels to scroll by
- +ve value indicates scrolling down
- -ve value indicates scrolling up

```
driver.execute_script("window.scrollBy(0, 500);")
```

3. Scrolling Horizontally:

- Similar to scrolling vertically
- The no. of pixels are provided as the first argument
- +ve value indicates scrolling to the right

```
driver.execute_script("window.scrollBy(500, 0);")
```

4. Scrolling to Page Height:

- Use the scrollTo method
- Pass the value document.body.scrollHeight in place of pixels as usual
- It refers to total height of the content in a webpage

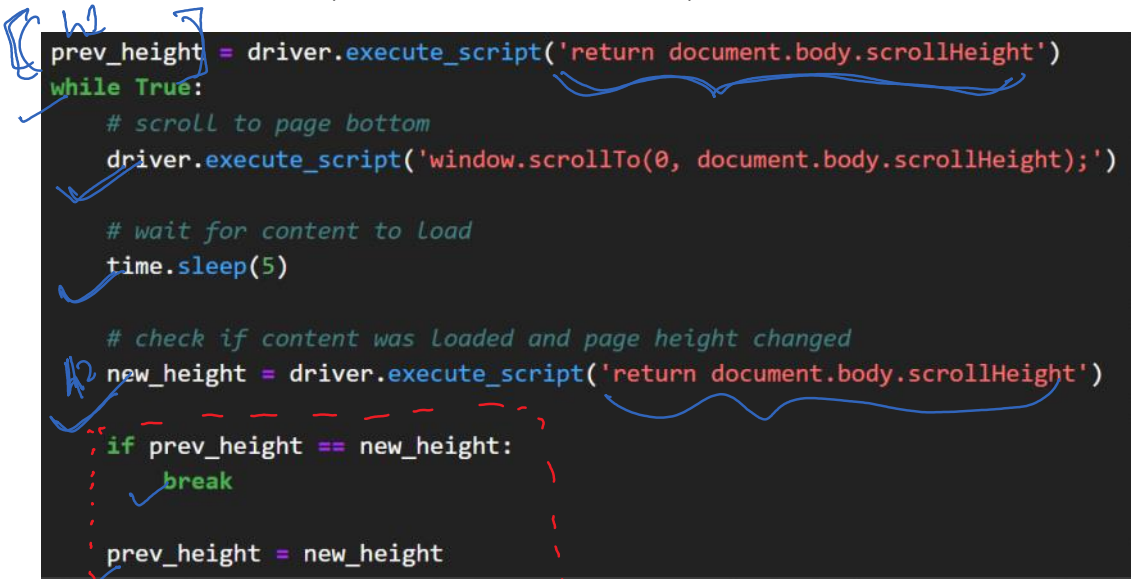
```
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
```

5. Infinite Scrolling:

- Initially, the webpage loads a fixed amount of content.
- As the user scrolls close to the bottom of the page, a JavaScript function triggers a request to load more content dynamically
- The new content is added to the page, and the process repeats.

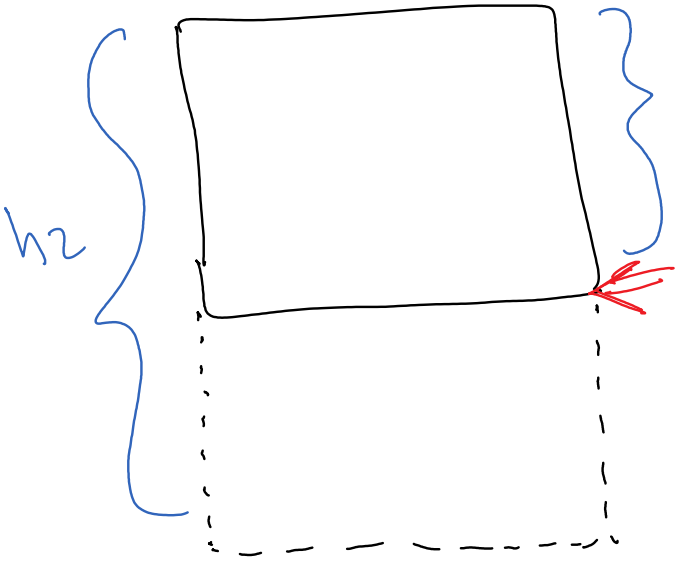
Algorithm:

- Get the height of the currently loaded page (h1)
- Run an infinite loop
- Scroll down the page to h1
- Inside the loop, get the height of the page again (h2)
- If h1 is same as h2, break out of the loop as no new content has been loaded
- If h1 is not same as h2, update h1 as h2 and continue the loop



```
prev_height = driver.execute_script('return document.body.scrollHeight')
while True:
    # scroll to page bottom
    driver.execute_script('window.scrollTo(0, document.body.scrollHeight);')

    # wait for content to load
    time.sleep(5)

    # check if content was loaded and page height changed
    new_height = driver.execute_script('return document.body.scrollHeight')

    if prev_height == new_height:
        break

    prev_height = new_height
```

```
prev_height = new_height
```



curr_height = h1

new_height = h2

curr_height == new_height

true → break

else → curr_height = new_height

1. Intro

Thursday, November 7, 2024 10:37 PM

- This section covers advanced web interactions that go beyond basic navigation and element manipulation.
- By mastering these techniques, developers will be able to handle real-world web scraping and automation challenges, including interacting with dynamic content, handling alerts, and managing iframes.

Topics:

- *Explicit Waits*
- *Implicit Waits*
- *Handling Alerts*
- *Frames & iFrames*

2. Explicit Waits

Friday, December 6, 2024 1:38 AM

What is Explicit Wait?

- Type of wait that pauses the execution of the script until a specific condition is met or a specified maximum time is reached .
- Useful when dealing with dynamic web elements that take time to appear or become interactable on the page.
- Helps avoid exceptions such as NoSuchElementException or ElementNotInteractableException
- Improves script reliability by waiting only as long as necessary
- Optimizes test execution time compared to fixed delays (e.g., time.sleep())

How to Implement?

- Selenium provides the WebDriverWait class to implement explicit waits
- It works with expected conditions defined in the selenium.webdriver.support.expected_conditions module
- The script checks for the condition at regular intervals (default - 500ms) until it's met or the timeout occurs

Syntax:

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, timeout, poll_frequency=0.5, ignored_exceptions=None)
```

- **Parameters:**
  - **driver**: The WebDriver instance controlling the browser
  - **timeout**: Maximum time (in seconds) to wait for the condition to be met
  - **poll_frequency**: How often (in seconds) the condition is checked (default: 0.5 seconds)
  - **ignored_exceptions**: A tuple of exceptions to ignore while waiting (optional)

- WebDriverWait often works in conjunction with Expected Conditions (EC) to define what to wait for:

CONDITION	DESCRIPTION
presence_of_element_located	Waits for the element to be present in the DOM
visibility_of_element_located	Waits for the element to be visible on the page
element_to_be_clickable	Waits for the element to be visible and enabled (clickable)
text_to_be_present_in_element	Waits for specific text to appear inside an element
alert_is_present	Waits for a browser alert to appear
title_is	Waits for the page title to exactly match a given value
title_contains	Waits for the page title to contain specific text

3. Implicit Waits

Friday, December 6, 2024 1:38 AM

What is Implicit Wait?

- Implicit Waits are a mechanism in Selenium to specify a default waiting time for the WebDriver when searching for elements on a webpage.
- If an element is not immediately found, the WebDriver waits for the specified duration before throwing a `NoSuchElementException`.
- This type of wait applies globally to all element searches in the WebDriver instance.

How It Works:

- We can set up the waiting duration using the `implicitly_wait` method of the webdriver instance.
- Makes Selenium scripts resilient to minor delays in the loading of web elements caused by network speed, animations, or dynamic content.
- Once set, it applies to all `find_element` and `find_elements` methods for the lifetime of the WebDriver instance.
- If the element is found before the timeout period, the script proceeds immediately. Otherwise, it waits until the timeout is reached and raises an exception if the element is still not found.

```
driver.implicitly_wait()
```

```
Signature: driver.implicitly_wait(time_to_wait: float) -> None
Docstring:
Sets a sticky timeout to implicitly wait for an element to be found,
or a command to complete. This method only needs to be called one time
per session. To set the timeout for calls to execute_async_script, see
set_script_timeout.

:Args:
 - time_to_wait: Amount of time to wait (in seconds)
```

Advantages:

- **Simplicity**: Easy to implement and applies globally, avoiding repetitive waits for every element.
- **Resilience**: Handles minor delays in loading dynamically generated elements, reducing script failures.
- **Better Control**: Provides a default buffer for all element searches without the need for explicit handling.

Disadvantages:

- Since it applies globally, it may not suit situations where different elements require different wait times.
- Mixing implicit waits with explicit waits can cause unpredictable behavior, as implicit waits can interfere with explicit wait polling mechanisms.
- Implicit waits only handle element visibility or presence and cannot wait for specific conditions like page titles or JavaScript execution.

Best Practices:

- Use either implicit waits or explicit waits in your script, but not both, to avoid conflicts.
- Set reasonable timeout durations and not very high implicit wait times (e.g., 60 seconds) as it can unnecessarily delay script execution.
- Implicit waits are suitable for simple scripts without complex wait conditions.

Implicit vs Explicit Wait:

FEATURE	IMPLICIT WAIT	EXPLICIT WAIT
Scope	Applies globally to all element searches	Applies to specific elements or conditions
Flexibility	Cannot target specific conditions or elements	Can wait for custom conditions like visibility or clickability
Ease of Use	Easier to implement for simple scenarios	Requires additional code for condition handling
Performance	May introduce unnecessary delays globally	More efficient as it targets specific waits
Application	Better for static pages where elements are predictable	Better for more dynamic and complex websites where elements can be unpredictable

4. Frames & IFrames

Friday, December 6, 2024 1:38 AM

What is a Frame/IFrame?

- In web development, **frames** and **iframes** are HTML elements that allow you to embed one HTML document inside another.

Frame:

- Part of the <frame> and <frameset> HTML tags, which were used in early web development to divide the browser window into multiple sections, each capable of loading a separate HTML document.
- Now obsolete in HTML5, frames are rarely used. They were replaced by iframes and other modern layout techniques like CSS Grid and Flexbox.

Iframe (Inline Frame):

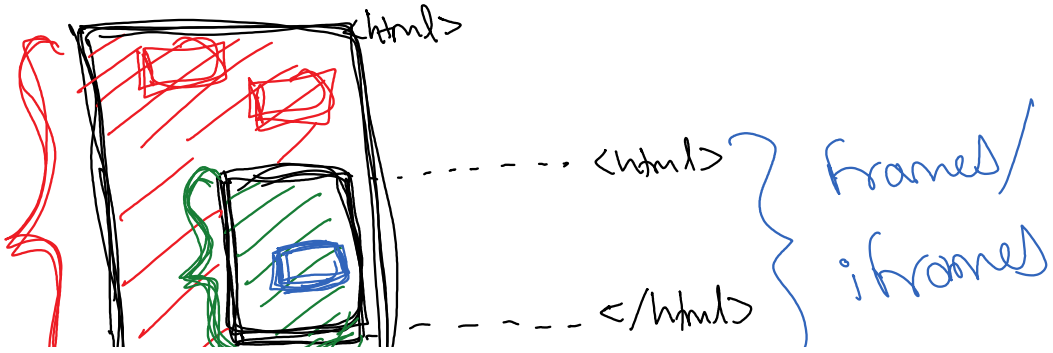
- An <iframe> is an HTML element that embeds another HTML document within the current page.
- Changes in the parent page (like CSS or JavaScript) typically do not affect the iframe's content, and vice versa.
- Each iframe has its own DOM (Document Object Model), CSS, and JavaScript scope.
- Interaction between the parent and iframe is restricted if they originate from different domains for security reasons.

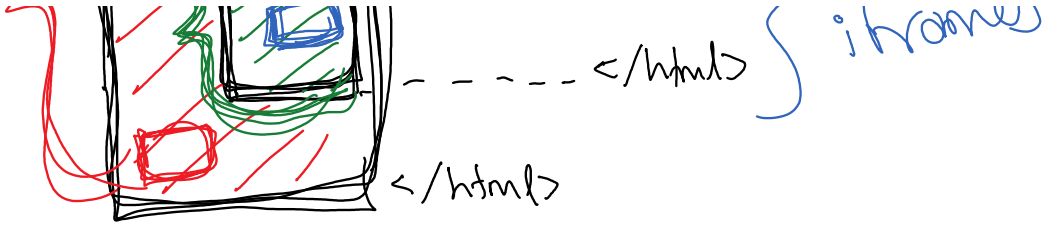
Working with Selenium:

- To interact with iframe content in Selenium, users must explicitly switch the context to the iframe using driver.switch_to.frame()
- Frame content can be identified in many different ways:
  - *ID*
  - *Name*
  - *Index*
  - *Xpath*
  - *CSS Selector*
- After interacting with an iframe, switch back to the parent page using driver.switch_to.default_content()

Best Practices:

- Ensure you know which frame or iframe contains the desired elements by inspecting the page source.
- Whenever possible, avoid switching by index to maintain flexibility if the page structure changes.
- Always switch back to the main content after interacting with a frame.




- - - - - - </html>
iframe
</html>

5. Handling Alerts

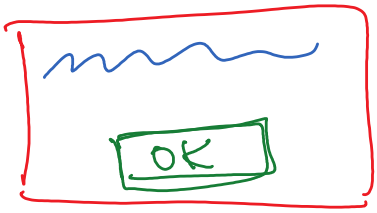
Friday, December 6, 2024 1:38 AM

What are Alerts:

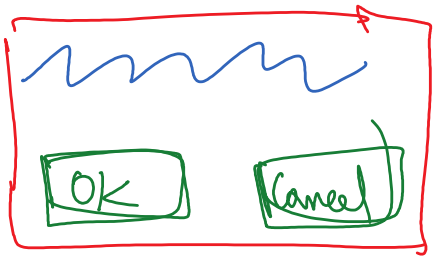
- Alerts in web refer to small, temporary messages or pop-ups that appear in a web browser to communicate information or request user actions.
- They are typically generated by JavaScript or built into the HTML/CSS structure of a webpage.
- Alerts are used for various purposes, including notifying users, obtaining confirmation, or prompting for input.

Types of Alerts:

1. **JavaScript alerts:**
   - Created using JavaScript's alert() function.
   - Displays a simple message to the user with a single "OK" button.
   - Blocks user interaction with the page until dismissed.



2. **Confirmation Alerts:**
   - Created using JavaScript's confirm() function.
   - Asks the user to confirm an action with "OK" and "Cancel" buttons.
   - Returns true for "OK" and false for "Cancel"



3. **Prompt Alerts**:
   - Created using JavaScript's prompt() function.
   - Requests user input and provides an input field along with "OK" and "Cancel" buttons.
   - Returns the input value for "OK" or null for "Cancel".

4. **Browser-Based Alerts (Authentication Pop-Ups)**:
   - Appear when a website requests HTTP basic authentication.
   - Requires entering a username and password

5. **Custom HTML Alerts (Modal Dialogs)**:
   - Designed using HTML, CSS, and JavaScript to create custom alert-style messages.
   - Offers more flexibility in design and functionality (e.g., styled dialog boxes with multiple buttons or inputs).

Handling Alerts with Selenium:

- To interact with alerts in Selenium, users must explicitly switch the context to the alert box using driver.switch_to.alert
- Use the accept() method to click the "OK" button
- Use the dismiss() method to click the "Cancel" button on confirmation pop-ups
- Use the text attribute to retrieve the message displayed on the alert
- Use the send_keys() method to input text into a prompt pop-up

1. Intro

Saturday, December 7, 2024 10:39 AM

- This section provides strategies for writing maintainable, efficient, and robust Selenium scripts.
- Adhering to these best practices will improve the performance of test automation or web scraping projects and make the code easier to debug and maintain.

2. Write Maintainable Code

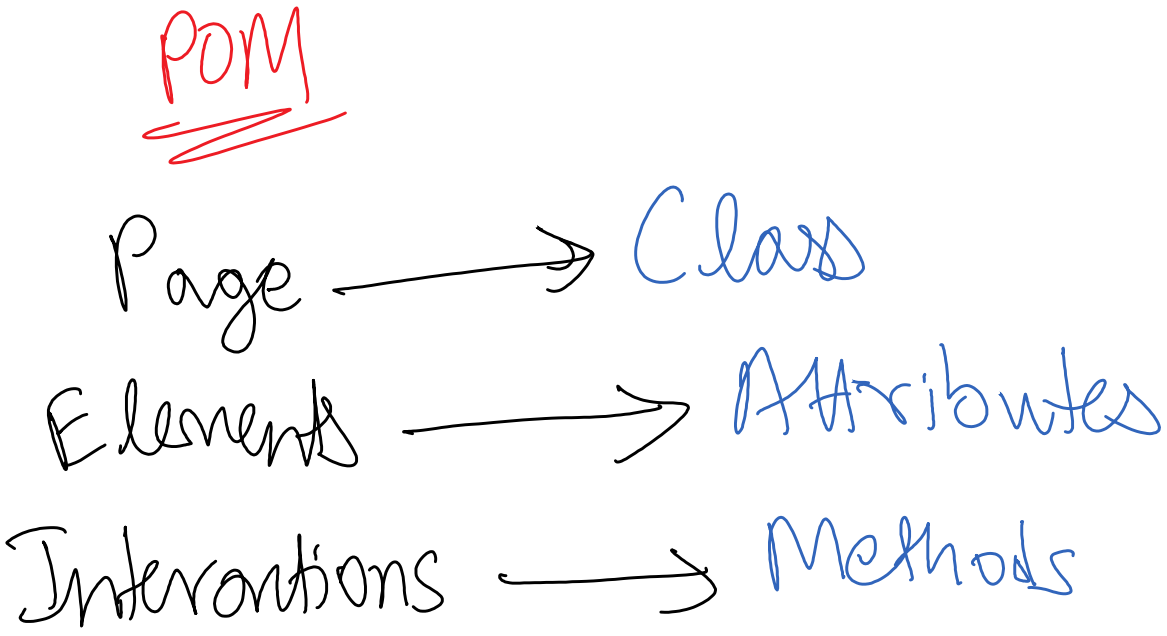
Saturday, December 7, 2024 10:41 AM

1. Page Object Model (POM):

- The Page Object Model is a design pattern that separates the code used to locate and interact with web elements.
- This improves code readability and reusability
- Each webpage is represented by a class
- Elements are defined as variables, and interactions (methods) are encapsulated within the class

2. Variable Names:

- Avoid overly generic names (ex: element1, button1)
- Use descriptive names to define web elements



3. Enhance Performance

Saturday, December 7, 2024 11:29 AM

1. Using Appropriate Waits:

- Overuse of time.sleep() can slow down scripts unnecessarily
- Look to use Explicit Waits and Implicit Waits for better performance
- Use Explicit Waits for better control

2. Optimize Locator Strategies:

- Use ID and NAME where possible as they are the fastest locators
- Avoid using XPath unless necessary, as it can be slower

3. Reuse Browser Sessions:

- Instead of launching a new browser for each test, reuse the browser session if applicable
- For scraping, minimize browser interaction by using headless mode

```
from selenium.webdriver.chrome.options import Options

options = Options()
options.add_argument("--headless")
driver = webdriver.Chrome(options=options)
```

4. Robustness and Error Handling

Saturday, December 7, 2024 11:38 AM

1. Try-Except blocks:

- Wrap code for critical interactions within try-except blocks to manage unexpected failures
- Catch specific exceptions if possible

```
try:
    element = driver.find_element(By.ID, "non_existent_id")
except NoSuchElementException:
    print("Element not found")
```

2. Release Resources:

- Always close the browser session at the end of the script to free resources
- This can be achieved using driver.quit()

5. Logging and Debugging

Saturday, December 7, 2024 11:43 AM

1. Logging:

- Avoid printing directly to the console to debug; use logs instead
- Use python's logging module for better control

```
import logging

logging.basicConfig(level=logging.INFO)
logging.info("Test started")
```

2. Capture Screenshots:

- Save a screenshot for debugging If code fails
- Use the save_screenshot method of the webdriver instance

3. Developer Tools:

- Use the browser's Developer Tools to inspect element locators and understand dynamic content of the webpage
- Gives a better understanding of the website structure and its respective code

6. Security Considerations

Saturday, December 7, 2024 11:49 AM

1. Secure Sensitive Data:

- Avoid exposing credentials in scripts
- Use encrypted files or environment variables for storage

2. Respect Website Policies:

- Check the website's robots.txt and Terms of Service before scraping
- Be ethical in your automation and scraping practices

Agenda

Wednesday, December 18, 2024 11:43 PM

- Project 1 - Yahoo Finance Stocks
- Project 2 - 99acres Real Estate Data

Action Plan

Sunday, December 15, 2024 9:47 AM

Scraping the most active stocks from Yahoo Finance:

1. Visit the target URL: https://finance.yahoo.com/
2. Hover over the Markets menu
3. Click on Trending Tickers
4. Click on Most Active
5. Ensure to visit all pages of stocks
6. Scrape the necessary data

Prerequisites

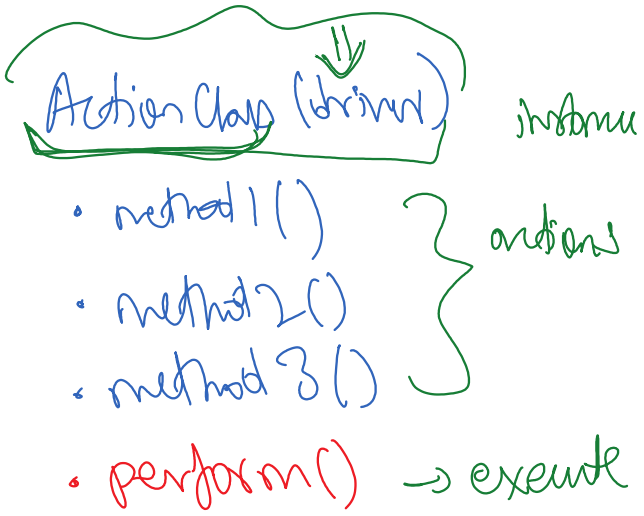
Tuesday, December 17, 2024 10:01 PM

1. Action Chains:

- Action Chains in Selenium is a feature that allows automating complex user interactions such as mouse and keyboard events.
- It is part of the ActionChains class in Selenium, designed to handle actions like clicking, dragging, hovering, and sending keyboard input.

Working of Action Chains:

- An instance of ActionChains is created by passing the WebDriver instance
- Use various methods provided by the class to define a sequence of actions
- Call the perform() method to execute all actions in the defined sequence



Common Methods:

- **click()**: Clicks on a specified web element
- **click_and_hold()**: Clicks without releasing the mouse button
- **double_click()**: Double-clicks on the element
- **context_click()**: Right-clicks on the element
- **move_to_element()**: Moves the mouse pointer to the element

2. Stocks Jargon:

a. Symbol:

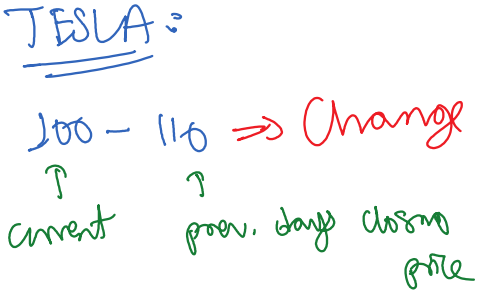
- This is a unique identifier for a particular stock listed on a stock exchange
- Ticker symbols are essential for identifying and trading stocks

b. Price:

- The current trading price of the stock (in dollars)
- This is the most recent price at which the stock was bought/sold on the market

c. Change:

- The difference in the stock's current price compared to the previous day's closing price
- +ve change indicates the price of the stock has increased from the previous day
- -ve change indicates the price of the stock has decreased from the previous day



d. Volume:

- The total number of shares of the stock during the current trading session
- Indicates the trading activity of the stock.
- Higher volume usually means more interest in the stock

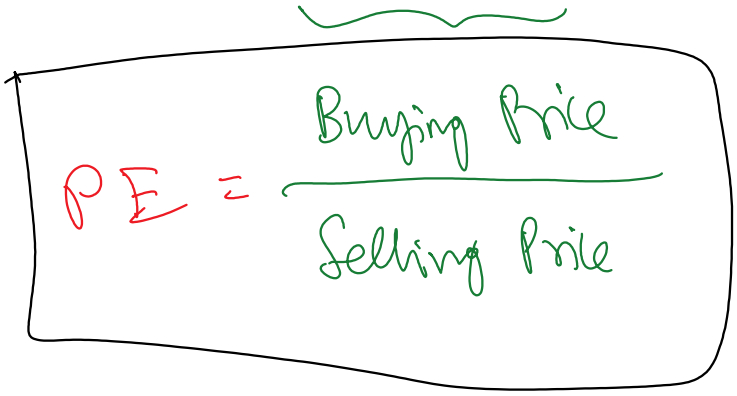
e. Market Cap:

- Represents the total value of a company in the stock market
- Ex: If a company has 1,000 shares and each costs $10, then the company's market value is $10,000

- The larger the cap, the more stable the company usually is

f. PE Ratio:

- Price to Earnings ratio
- Can be thought of as the ratio of the CP of a stock to the SP of the stock

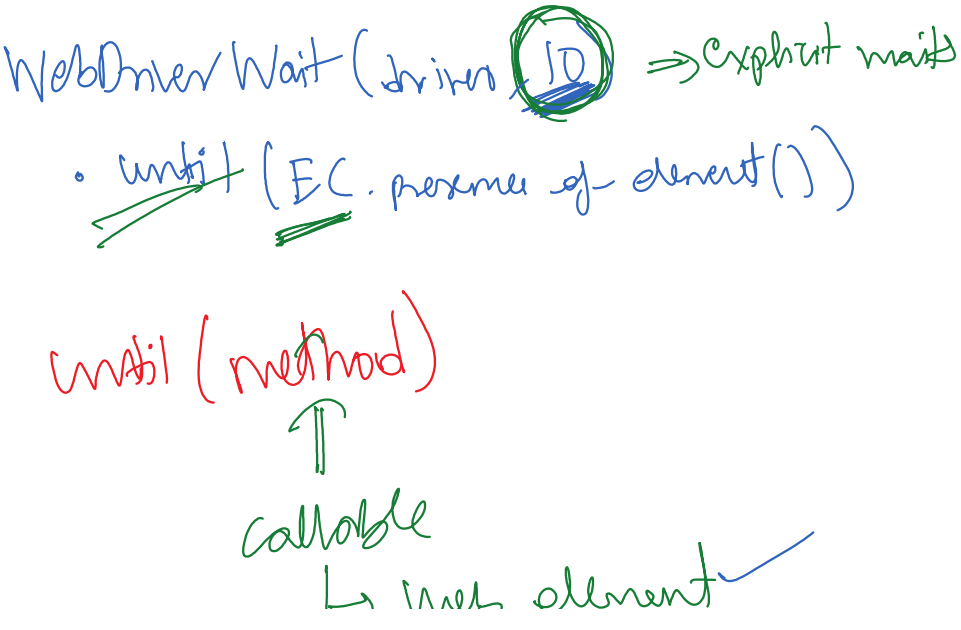


3. Loading a Webpage:

- We can use `return document.readyState` to retrieve the current loading state of the web page
- The values returned by this command can be:
  - loading: The document is still loading
  - interactive: The document has been loaded but external resources like images and stylesheets might still be loading
  - complete: The entire document, including external resources, has been fully loaded
- This is commonly used in Selenium scripts to wait until the page is fully loaded before interacting with elements

4. until():

- It has a parameter method
- This parameter must be a callable (function or lambda) which takes a WebElement as input and returns a boolean value
- If the callable doesn't return a truthy value within the timeout period, a TimeoutException exception will be raised



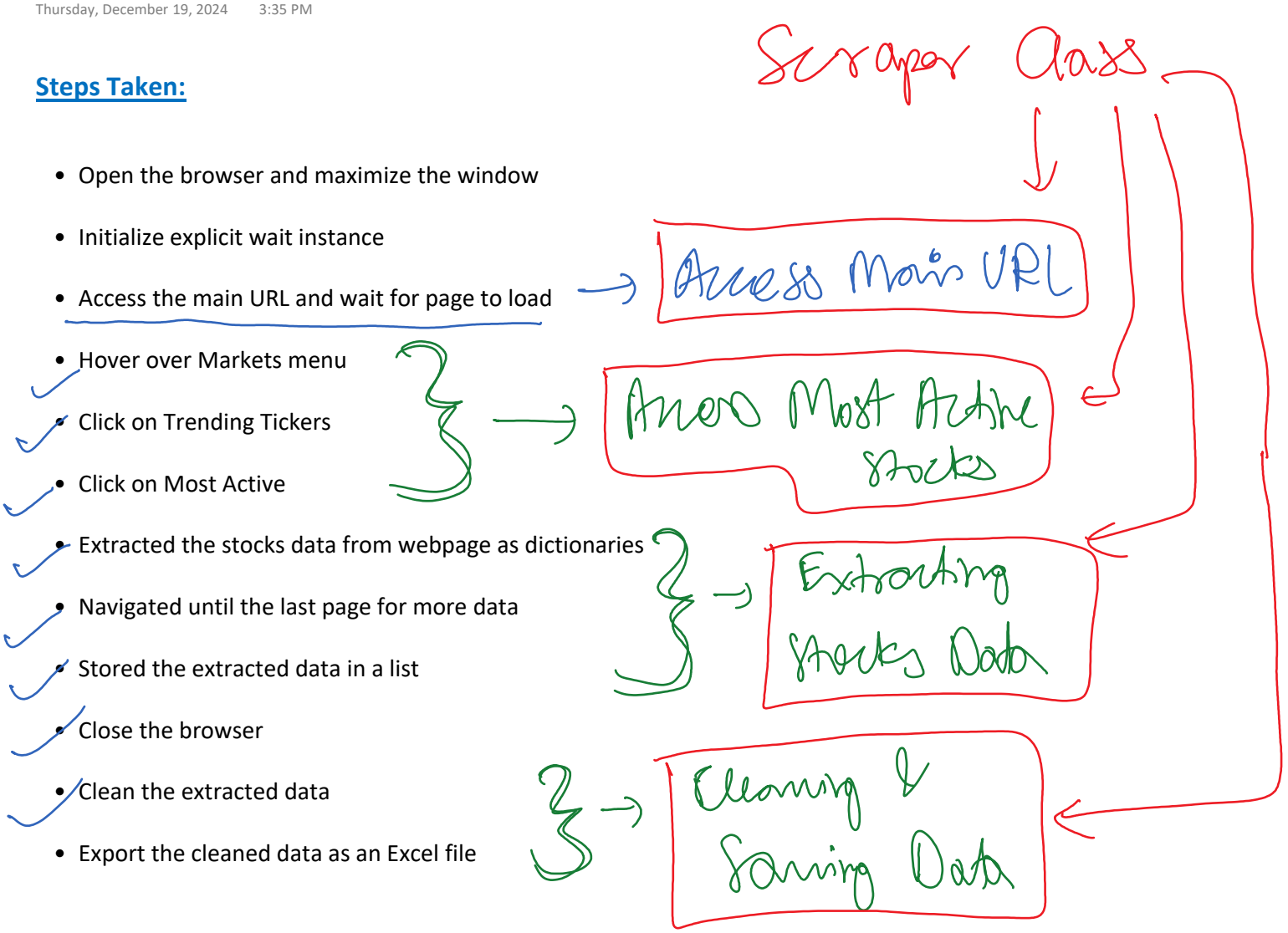
collapse
- web element
- return boolean

Code Restructuring

Thursday, December 19, 2024 3:35 PM

Steps Taken:

- Open the browser and maximize the window
- Initialize explicit wait instance
- Access the main URL and wait for page to load
- Hover over Markets menu
- Click on Trending Tickers
- Click on Most Active
- Extracted the stocks data from webpage as dictionaries
- Navigated until the last page for more data
- Stored the extracted data in a list
- Close the browser
- Clean the extracted data
- Export the cleaned data as an Excel file

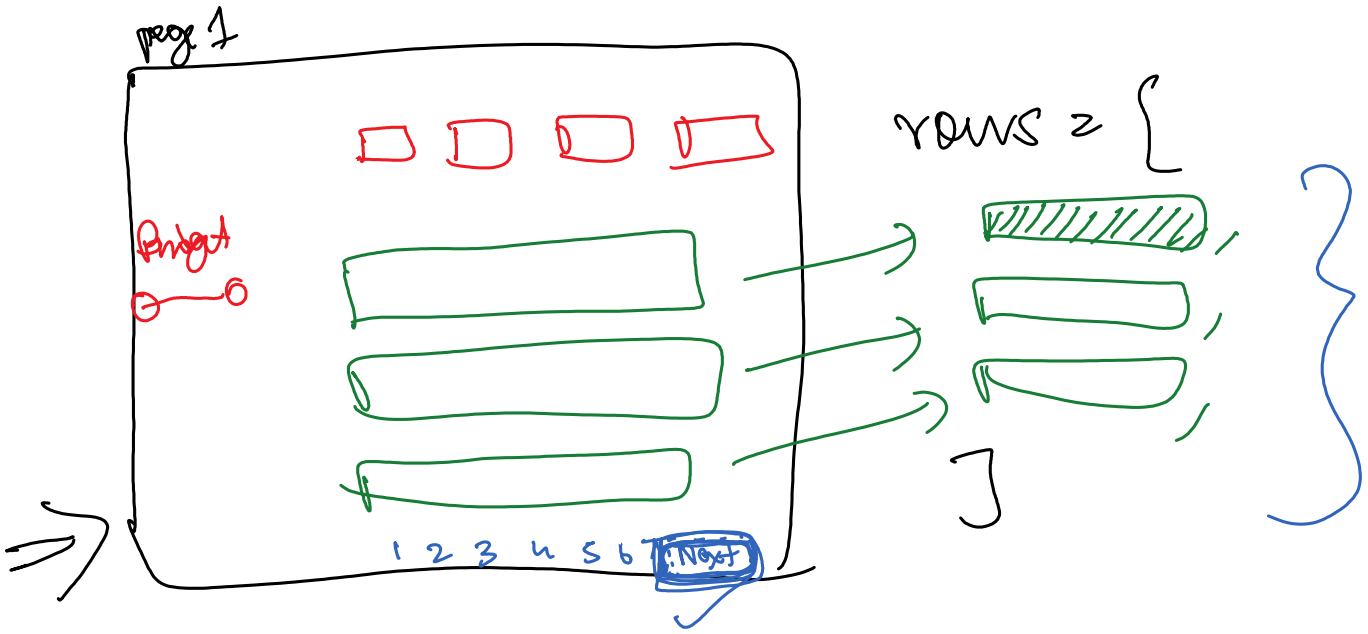


Action Plan

Tuesday, December 24, 2024 12:11 AM

Scraping the real-estate listings from 99acres:

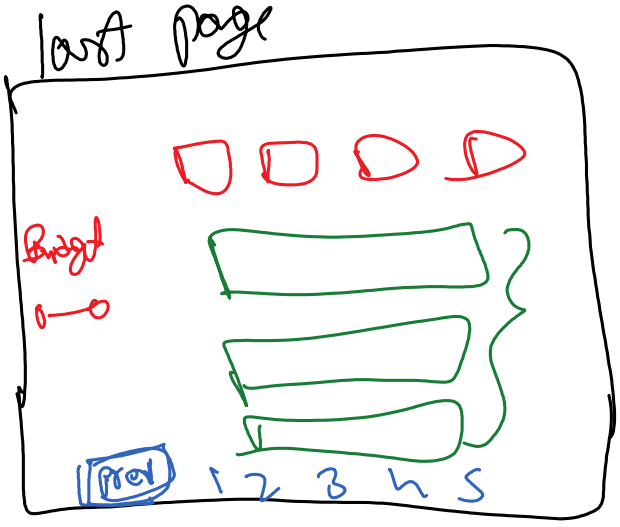
1. Set browser Options before instantiating the web driver
2. Visit the target URL: https://www.99acres.com/
3. Type Chennai in the search bar
3. Click on Search button
4. Adjust the Budget slider to 5cr INR
5. Click on the below options to filter the resulting data:
    - *Verified*
    - *Ready To Move*
    - *With Photos*
    - *With Videos*
6. Scrape the necessary data
7. Clean the extracted data
8. Export cleaned data as an excel sheet



Inside page: (2)

1st to 2nd last page

1. Identify "next page" button ✓
2. Scroll down to it ✓
3. Extract page data ✓
4. Click on "next page" button ✓
5. Wait 5s ✓

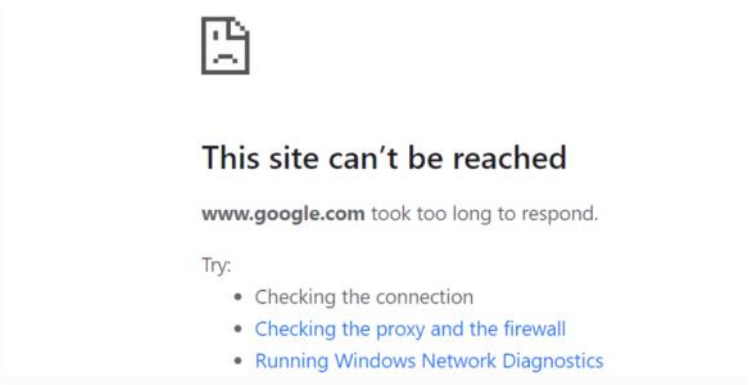


1. Try to identify "next page" btn
2. Break out of loop

Prerequisites

Tuesday, December 24, 2024 10:46 PM

1. "This site can't be reached":



- **Causes:**
  - The ChromeDriver or WebDriver version might not match the installed browser version
  - A proxy or VPN might interfere with the request
  - Firewall or corporate network restrictions may block access
  - Websites often restrict access to bots or automated tools like Selenium
  - The website might have anti-scraping mechanisms in place
  - The certificate or SSL/TLS version might not be supported or configured correctly

- **Solutions:**
  - Ensure your WebDriver matches the browser version
  - Configure browser options in Selenium
  - Force the browser to use HTTP/1.1
  - Use a user-agent string to make requests appear like they are coming from a browser
  - Run the browser in incognito mode

2. Selenium Options:

- In Selenium, Options is used to customize and configure browser settings and behavior when automating browsers
- Allows to specify preferences, enable or disable features, and set options that are specific to the browser you are automating
- Each browser driver has its own Options class to provide these configurations

Common Usecases:

- Running browser in headless mode (without GUI)
- Disabling browser notifications
- Setting the default download directory

- Adding browser extensions

Key Methods:

- **add_argument(arg)**: Adds a command-line argument to the browser
- **add_experimental_option(name, value)**: Adds experimental options or preferences
- **set_capability(name, value)**: Sets desired capabilities for the browser

Advantages:

- Helps customize browser behavior according to your testing needs
- Suppress unnecessary UI elements like notifications or pop-ups
- Ensure the browser runs with specific settings each time, promoting consistency
- Improves efficiency as we can run browsers in headless mode to save resources during testing

3. getBoundingClientRect():

- JavaScript function, used to manipulate and interact with elements on a webpage by executing JavaScript
- useful when standard Selenium methods can't achieve certain tasks directly, such as precise scrolling or positioning
- This approach ensures that Selenium scripts can interact with dynamically loaded or partially visible elements more reliably

Advantages:

- Helps in cases where the elements are not immediately visible in the viewport
- Handles scenarios where the webpage layout changes, requiring precise adjustments
- Ensures elements are well-placed for user interaction or screenshots
- Works even when the standard Selenium methods (scrollIntoView(), actions.move_to_element(), etc.) don't work as intended