

By Pranit Dalvi

Introduction

This documentation covers various topics related to Web Scrapping through python.

* **Installation**: The documentation begins with how to install the Selenium library for your chosen programming language (e.g., Python, Java, C#). You'll use a package manager like pip for Python: pip install selenium.
* **WebDriver**: This is the core of Selenium. It's an API that allows you to control a web browser. The documentation covers how to download and configure the specific WebDriver for the browser you want to automate (e.g., ChromeDriver, GeckoDriver for Firefox, MSEdgeDriver).

**Navigating Web Pages**

* **Launching a Browser**: The documentation shows you how to import WebDriver and create an instance of a browser. For example, in Python: from selenium import webdriver and driver = webdriver.Chrome().
* **Opening a URL**: You'll learn how to navigate to a specific web page using the get() method: driver.get("https://www.google.com").
* **Browser Control**: This section explains how to manage browser windows, such as maximizing the window (driver.maximize\_window()) or closing it (driver.close()).

**Locating Elements**

This is a crucial part of Selenium, and the documentation provides detailed examples for each method. You need to tell Selenium which element on the page you want to interact with.

* **find\_element(By.ID, "elementId")**: Locates an element by its unique id attribute.
* **find\_element(By.NAME, "elementName")**: Finds an element by its name attribute.
* **find\_element(By.CLASS\_NAME, "className")**: Locates an element by its class attribute.
* **find\_element(By.TAG\_NAME, "tagName")**: Finds an element by its HTML tag name (e.g., <div>, <a>).
* **find\_element(By.LINK\_TEXT, "Link Text")**: Used to locate a hyperlink by the exact text it displays.
* **find\_element(By.PARTIAL\_LINK\_TEXT, "Partial Link Text")**: Useful for finding a hyperlink when you only know part of its text.
* **find\_element(By.CSS\_SELECTOR, "cssSelector")**: A powerful way to find elements using CSS selectors, which are a fast and reliable method.
* **find\_element(By.XPATH, "xpathExpression")**: Locates an element using an XPath expression, which is very flexible and can be used to navigate complex or dynamic web pages.

**Interacting with Elements**

Once you've located an element, the documentation shows you how to interact with it.

* **click()**: Simulates a mouse click on the element.
* **send\_keys("text")**: Enters text into a text field or text area.
* **clear()**: Clears the content of a text input field.
* **submit()**: Submits a form.

**Handling Waits**

Modern websites are dynamic, and elements might not be immediately available. The documentation covers different types of waits to handle this.

* **Implicit Wait**: This sets a timeout for all find\_element calls. Selenium will wait for a certain duration before throwing an exception if an element is not found.
* **Explicit Wait**: This is a more specific wait. You tell Selenium to wait for a particular condition to be met before proceeding, such as an element becoming clickable or visible.

**Additional Topics**

* **Handling Alerts**: How to accept or dismiss pop-up alerts.
* **Handling Frames and Windows**: How to switch control between different frames (<iframe>) or new browser tabs/windows.
* **Taking Screenshots**: Capturing a screenshot of the current browser window.
* **Cookies**: How to manage browser cookies.
* **Headless Browsers**: Running a browser in the background without a graphical user interface. This is common for automation on servers.