**Selenium:**

Selenium is an open-source suite of tools and libraries that is used for [browser automation](https://www.browserstack.com/guide/what-is-browser-automation). Selenium us used to

* It allows users to test their websites functionally on different browsers.
* Perform [Cross browser testing](https://www.browserstack.com/cross-browser-testing) to check if the website functions consistently across different browsers.

It provides a single interface that lets you write test scripts in programming languages like Ruby, Java, NodeJS, PHP, Perl, Python, JavaScript, and C#, among others. Selenium is very extensible and can be integrated with other tools and frameworks like TestNG, JUnit, Cucumber, etc.

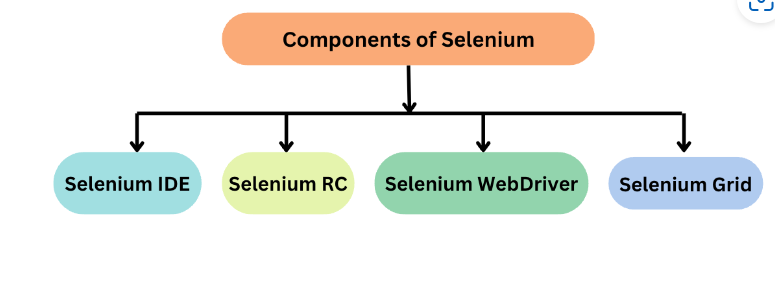
**Importance of Testing in Selenium:**

Manual testing can be time-consuming and prone to human errors. [Selenium Automation](https://www.browserstack.com/guide/selenium-framework) allows tests to be executed quickly and accurately, reducing the likelihood of human mistakes and ensuring consistent test results.

Selenium allows developers and testers to automate the testing of web applications across different browsers and platforms.

* **Language Support:** Selenium allows you to [create test scripts](https://www.browserstack.com/guide/run-selenium-test-script) in different languages like [Ruby](https://www.browserstack.com/guide/selenium-ruby-tutorial), [Java](https://www.browserstack.com/guide/selenium-with-java-for-automated-test), [PHP](https://www.browserstack.com/guide/selenium-and-php-tutorial), Perl, [Python](https://www.browserstack.com/guide/automate-with-selenium-python), [JavaScript](https://www.browserstack.com/guide/automation-using-selenium-javascript), and [C#](https://www.browserstack.com/guide/selenium-with-c-sharp-for-automated-test), among others.
* **Browser Support:** Selenium enables you to test your website on different browsers such as [Google Chrome](https://www.browserstack.com/guide/run-selenium-tests-using-selenium-chromedriver), [Mozilla Firefox](https://www.browserstack.com/guide/run-selenium-tests-using-firefox-driver), [Microsoft Edge](https://www.browserstack.com/guide/launch-edge-browser-in-selenium), [Safari](https://www.browserstack.com/guide/run-selenium-tests-on-safari-using-safaridriver), [Internet Explorer (IE)](https://www.browserstack.com/guide/run-selenium-tests-using-ie-driver), etc.
* **Scalability:** Automated testing with Selenium can easily scale to cover a wide range of [test cases](https://www.browserstack.com/guide/how-to-create-selenium-test-cases), scenarios, and user interactions. This scalability ensures [maximum test coverage](https://www.browserstack.com/guide/how-to-ensure-test-coverage) of the application’s functionality.
* **Reusable Test Scripts:** Selenium allows testers to create reusable test scripts that can be used across different test cases and projects. This reusability saves time and effort in test script creation and maintenance.
* [**Parallel Testing**](https://www.browserstack.com/guide/what-is-parallel-testing): Selenium supports parallel test execution, **allowing multiple tests to run concurrently**. This helps reduce the overall testing time, making the development process more efficient.
* **Documentation and Reporting:** Selenium provides detailed test execution logs and reports, making it easier to track test results and identify areas that require attention.
* **User Experience Testing:** Selenium can simulate user interactions and behavior, allowing testers to assess the user experience and ensure that the application is intuitive and user-friendly.
* **Continuous Integration and Continuous Deployment (CI/CD):** Selenium can be integrated into CI/CD pipelines to automate the testing of each code change. This integration helps identify and address issues earlier in the development cycle, allowing for faster and more reliable releases

## Components of Selenium



Selenium Suite has 4 components namely:

1. Selenium IDE
2. Selenium RC
3. Selenium WebDriver
4. Selenium Grid

### Selenium IDE:

[Selenium IDE](https://www.browserstack.com/guide/what-is-selenium-ide) is a Chrome and Firefox plugin. The primary use of a Selenium IDE is to record user interactions such as clicks, selections etc in the browser and plays them back as automated tests .

It then generates the test script (of the automated tests) in programming languages like C#, Java, Python, and Ruby and Selenese (Selenium’s own scripting language).

Selenium IDE helps in:

* Creating automated test scripts and validating them at speed
* Identifying and highlighting errors during the replay of interactions
* Cross Browser Testing

**Selenium RC** :

Selenium RC was built to automate the testing of web applications by simulating user interactions across different browsers and platforms. **It provided a way to** **browser automation remotely and execute test scripts written in various programming languages.**

**Limitations of Selenium RC:**

* **Browser Limitations:** Selenium RC had to work with browsers using a JavaScript-based “proxy” mechanism, which introduced potential instability and limitations, especially when working with modern web applications.
* **Speed and Performance**: The use of a JavaScript proxy added overhead and affected the speed and performance of test execution.
* **Maintenance and Compatibility:** Selenium RC required separate “drivers” for each browser, making maintenance and compatibility challenging as browsers continued to update and evolve.
* **Synchronization Issues:** Selenium RC often faced synchronization problems, where test scripts had to wait for the browser to respond before proceeding to the next step.
* **Complex Setup**: Setting up Selenium RC involved multiple components, which could be complex and difficult to configure correctly.

**Selenium WebDriver:**

[Selenium WebDriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial) is a powerful and enhanced version of Selenium RC which was developed to overcome the limitations of Selenium RC. WebDriver communicates with browsers directly with the help of browser-specific native methods, thereby completely eliminating the need of Selenium RC.

WebDriver works closely with Selenium IDE and Selenium Grid resulting in reliable test execution at speed and scale.

**Selenium Grid:**

[Selenium Grid](https://www.browserstack.com/guide/selenium-grid-tutorial) is a smart proxy server that allows QAs to run tests in parallel on multiple machines. This is done by routing commands to remote web browser instances, where one server acts as the hub. This hub routes test commands that are in JSON format to multiple registered Grid nodes.

## Understanding Selenium WebDriver

### What is Selenium WebDriver:

Selenium WebDriver is an open source tool to perform Browser Automation on real browsers. WebDriver communicates with browsers directly using client libraries and JSON wire protocol. It helps testers ensure that the website functions as intended on different browsers.

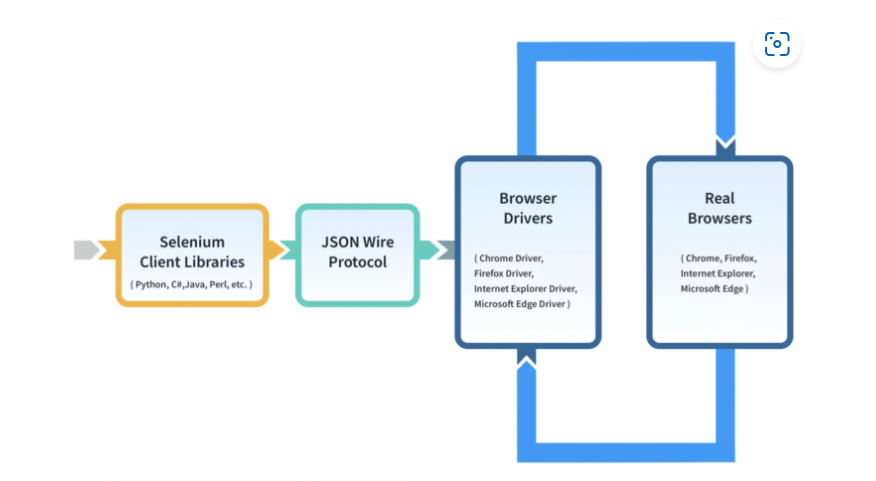
### When to use Selenium WebDriver

Selenium WebDriver is primarily designed to test web applications. You can perform:

* [**Functional Testing**](https://www.browserstack.com/guide/functional-testing)**:** Test the functionality of web applications by automating user interactions such as clicking buttons, filling out forms, navigating pages, and verifying expected outcomes.
* [**Cross-Browser Testing**](https://www.browserstack.com/cross-browser-testing)**:**Test web application for consistency across different browsers and browser versions (e.g., Chrome, Firefox, Edge, Safari)
* [**Cross-Platform Testing**](https://www.browserstack.com/guide/cross-platform-testing)**:** Test the web application on different operating systems sucn as Windows, macOS, and Linux.
* [**Regression Testing**](https://www.browserstack.com/guide/regression-testing)**:** Test the existing functionality of the web app, whenever a new feature is introduced.
* [**Data-Driven Testing**](https://www.browserstack.com/guide/data-driven-framework-in-selenium): Execute the same test **script with different sets of input data to** validate different scenarios.
* [**Parallel Testing**](https://www.browserstack.com/guide/what-is-parallel-testing)**:** Selenium WebDriver can be integrated with tools like Selenium Grid to execute tests in parallel across multiple browsers and platforms, reducing testing time.
* **Complex User Flows:** Simulate complex user workflows or interactions that need to be tested repeatedly.
* **Integration with Continuous Integration (CI) Pipelines:** Selenium WebDriver can be seamlessly integrated into your [CI/CD pipeline](https://www.browserstack.com/guide/building-ci-cd-pipeline), enabling automated testing with every code change and ensuring code quality.
* [**UI/UX Testing**](https://www.browserstack.com/guide/ui-testing-guide)**:** Automate user interface (UI) testing, ensuring that the visual elements and layout of your application are consistent.
* [**End-to-End Testing**](https://www.browserstack.com/guide/end-to-end-testing)**:** Selenium WebDriver can be used for end-to-end testing, where you simulate real user interactions across different parts of the application to ensure a seamless user experience.
* [**Page Object Model (POM)**](https://www.browserstack.com/guide/page-object-model-in-selenium)**Implementation:**If you adopt the Page Object Model (POM) design pattern, WebDriver can be used effectively to encapsulate the interactions and elements of different web pages, leading to more maintainable and modular test scripts.
* **Complex Scenarios:** When dealing with complex scenarios such as [handling alerts, pop-ups](https://www.browserstack.com/guide/alerts-and-popups-in-selenium), [iframes](https://www.browserstack.com/guide/handling-frames-in-selenium" \t "_blank), and dynamic content, Selenium WebDriver provides the flexibility to address these challenges.
* [**Performance Testing**](https://www.browserstack.com/guide/performance-testing)**:**While not its primary use, WebDriver can be used to simulate user load and interactions for basic performance testing.

### Architecture of Selenium WebDriver (Selenium 3):

WebDriver Architecture is made up of four major components:



* **Selenium Client library:** Selenium provides support to multiple libraries such as Ruby, Python, Java, etc as language bindings
* **JSON wire protocol over HTTP:** JSON is an acronym for JavaScript Object Notation. It is an open standard that provides a transport mechanism for transferring data between client and server on the web.
* **Browser Drivers:** Selenium browser drivers are native to each browser, interacting with the browser by establishing a secure connection. Selenium supports different browser drivers such as ChromeDriver, GeckoDriver, Microsoft Edge WebDriver, SafariDriver, and InternetExplorerDriver.
* **Browsers:** Selenium provides support for multiple browsers like Chrome, Firefox, Safari, Internet Explorer etc.

JSON wire protocol helps Selenium Language Bindings communicate with the Browser Drivers and thus automate interactions on real browsers.

### Browsers Supported in Selenium WebDriver

Selenium supports these Browsers:

* Google Chrome
* Mozilla Firefox
* Safari
* Internet Explorer
* Microsoft Edge

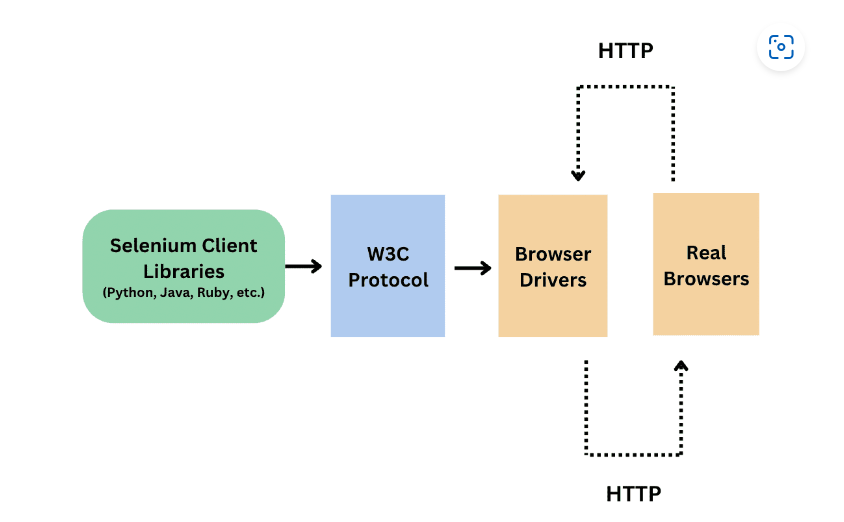
Note: Selenium 4 has terminated native support to Opera and PhantomJS browsers

## Understanding Selenium 4

Selenium 4 is an advanced version of Selenium and was introduced to improve the efficiency, performance and capabilities of browser automation.

### Architecture of Selenium 4

The architecture of Selenium 4 is similar to Selenium 3, however it uses W3C protocol instead of JSON wire protocol for communication between Client Libraries and Browser Drivers.



W3C protocol was introduced because all the web browsers followed the W3C standards and also all the browser drivers followed the W3C standards. To standardise the communication, JSON wire protocol was replaced by W3C in Selenium 4.

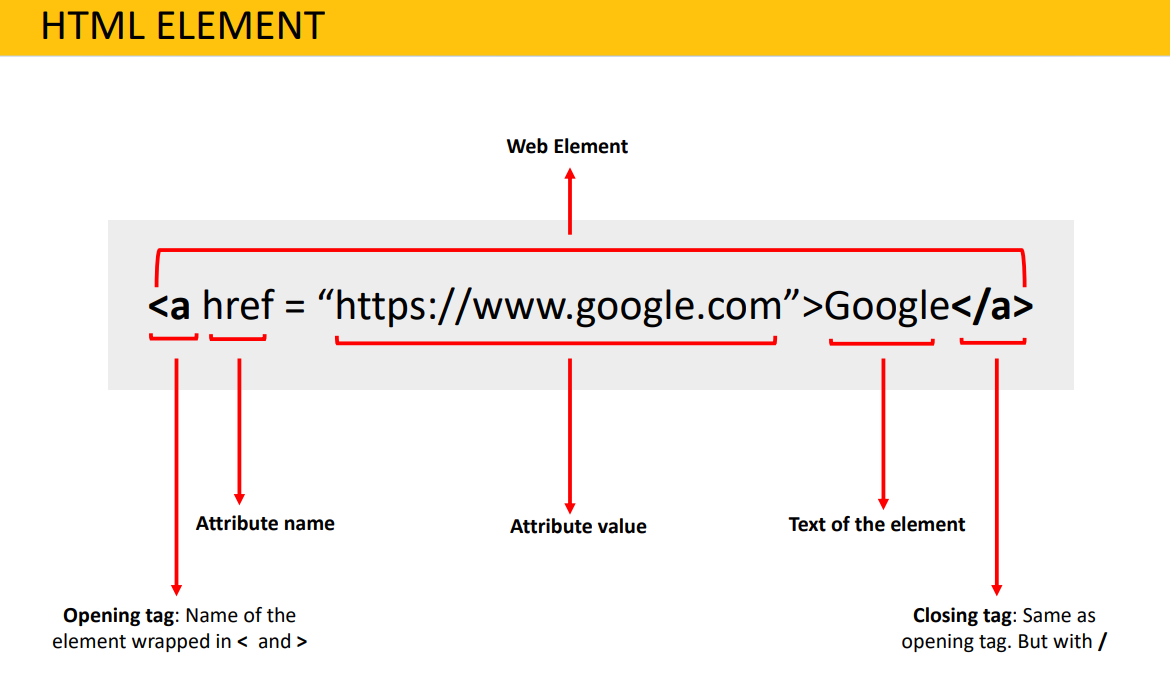
This helped in better communication with the browsers, stability, and common code (i.e. no browser specific code required). Due to W3C there is a direct transfer of information between client and server.

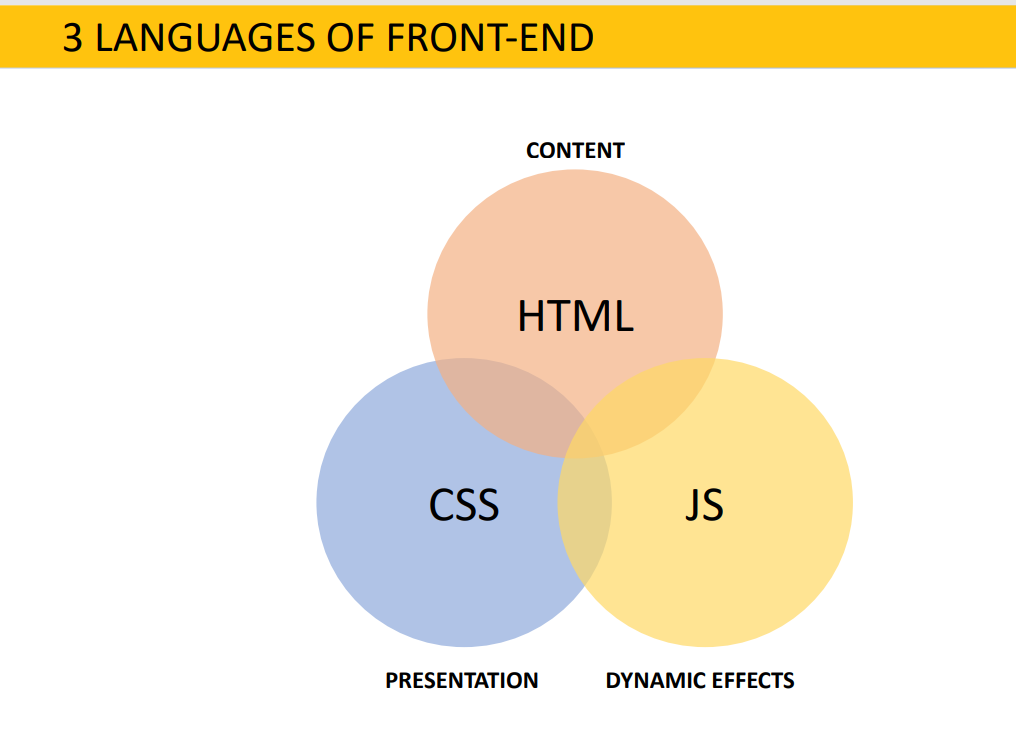
### Feature Highlights of Selenium 4

* Terminated native support to Opera and PhantomJS browsers
* Selenium 4 introduced relative locators (enable you to locate web elements on a page based on their relationship to other elements)
* Deprecated Desired Capabilities and replaced it with Options class
* Added new methods to Action class

**HTML Basics:**

* Hyper Text Markup Language HTML is a markup language that web developers use to structure and design the content of a webpage (HTML is not a programming language)
* HTML consists of elements (web elements) that describe different types of contents like links, headings, images, text boxes, radio buttons, check-boxes etc.
* Browser understands HTML and renders HTML code as websites





**Launching different browsers:**

**Note: download the drivers from different vendors and store it in executable folder**

from selenium import webdriver  
driver = webdriver.Chrome()

from selenium import webdriver  
driver = webdriver.Edge()

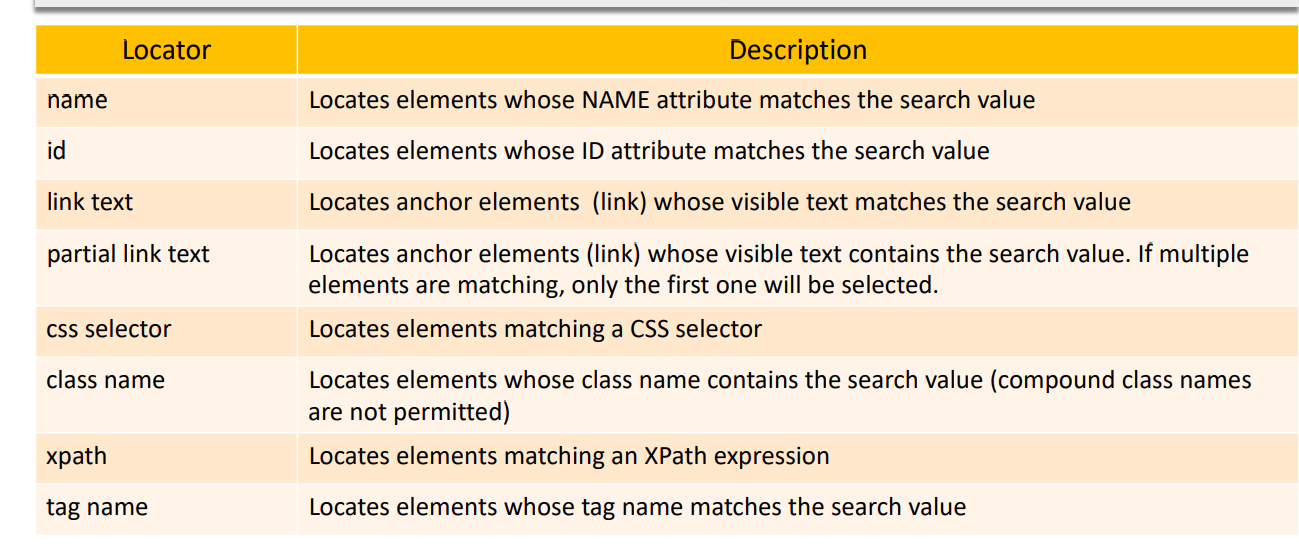
**Common browser actions:**

from selenium import webdriver  
driver = webdriver.Edge()  
driver.get(r"https://www.postman.com")  
driver.maximize\_window()  
driver.minimize\_window()  
print(driver.current\_url)  
print(driver.title)  
driver.close()  
driver.quit()

**Locators:**

Locators are used to find the webelement in the webpage using few inbuilt methods provided by selenium

There are 8 different built-in element location strategies in Webdriver



**Methods to find the webelements using locators:**

* Find\_element()
* Find\_elements()

**find\_element():**

* find\_element method returns a web element if the element is found in the DOM/HTML.
* If no element is found, find\_element method throws **"NoSuchElementException"** If there are multiple elements matching the same locator, find\_element method returns the first matching element from the DOM

we can find the one webelement at a time using this, before using this we need to import By class

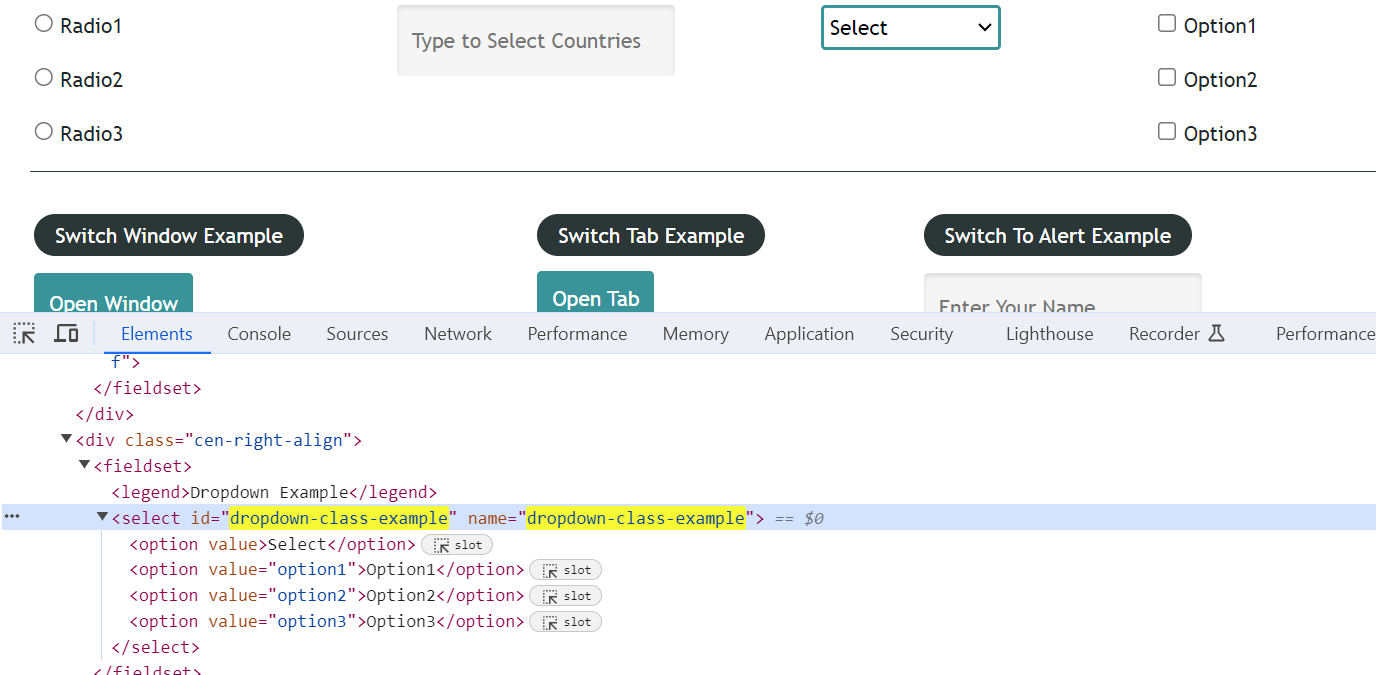
After finding the web element, if you want to perform the click operation you need to use .clikc() followed by driver.find() and if it is a text filed if you want to enter the text you need to use .send\_keys(‘text to pass”)

**find\_elements():**

* Returns list of web elements. Each item of the list is a web element.
* If there are no web elements which matches the given property and its value, find\_elements method returns an empty list

**Different locators:**

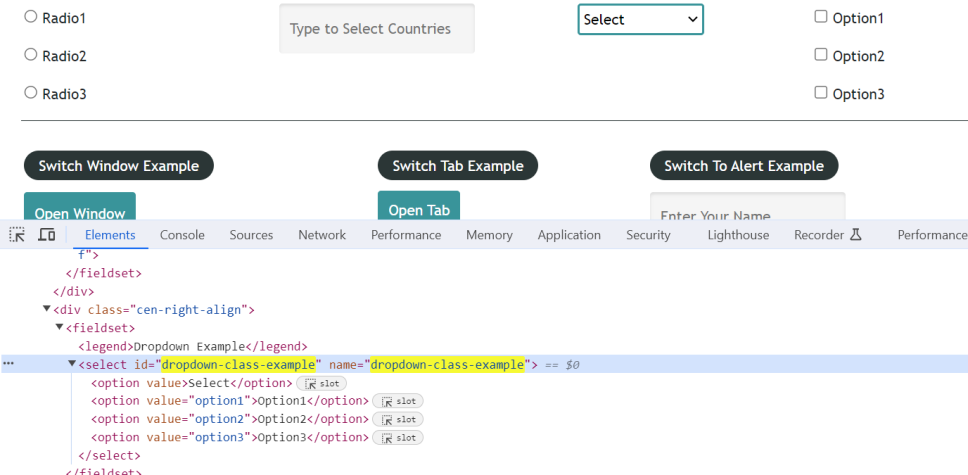
**#using name**

****

driver.find\_element(By.NAME, "dropdown-class-example").click()

driver.find\_element(By.NAME, "enter-name").send\_keys("Moahan kasi")

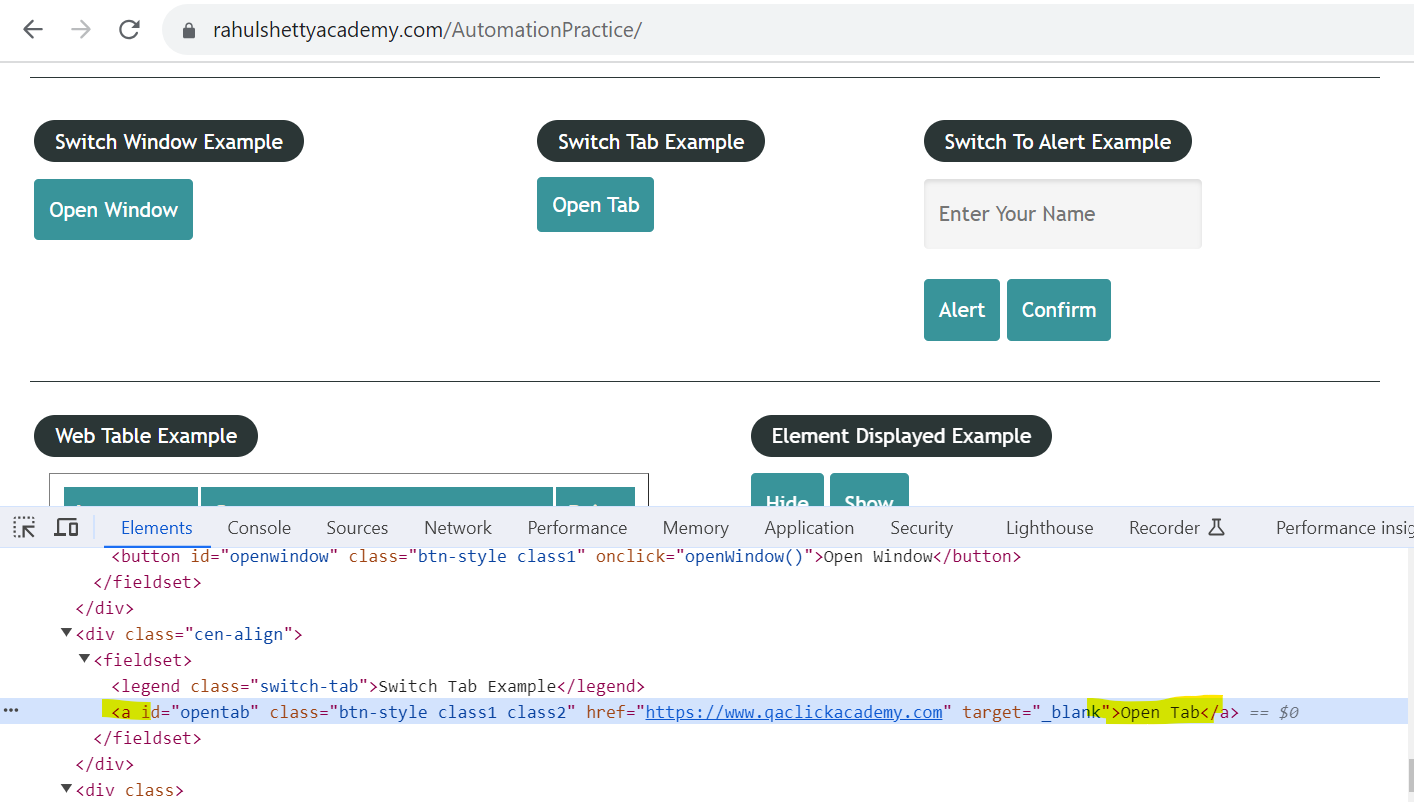
#usign id



driver.find\_element(By.ID, "checkBoxOption1").click()

driver.find\_element(By.ID, "autocomplete").send\_keys("Brazil")

#link text



**driver.find\_element(By.LINK\_TEXT, "Open Tab").click()**

Note: here we will provide the text of the link tag field

**Patial link text:**



**driver.find\_element(By.PARTIAL\_LINK\_TEXT, "Bad").click()**

Note: here we will enter the few text of link tag filed

CSS Selector:

CSS (Cascading Style Sheets) Selectors in Selenium are used to identify and locate web elements based on their id, class, name, attributes and other attributes. CSS is a preferred locator strategy as it is simpler to write and faster as compared to XPath.

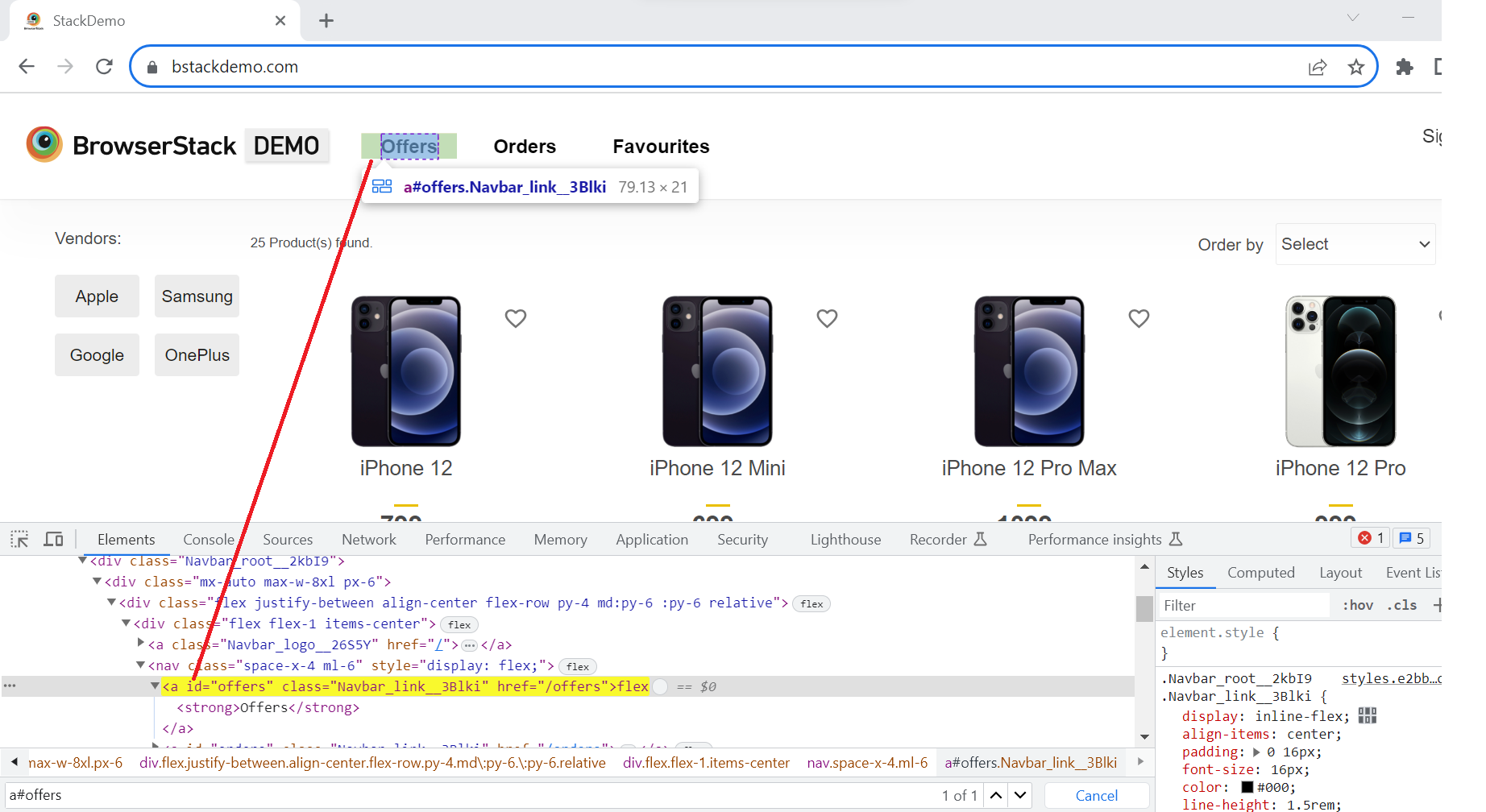
Types of CSS Selectors in Selenium (with Examples)

There are five types of CSS Selectors in Selenium tests:

* **ID**
* **Class**
* **Attribute**
* **Sub-String**
* **Inner String**

**ID:**

**In CSS, we can use “#” notation to select the “id” attribute of an element.**



Syntax:

driver.findElement(By.cssSelector(“<tagname>#<id value>”));

driver.findElement(By.cssSelector(“#<id value>”));

driver.findElement(By.cssSelector(“<tagname>[id=’<id value>’]”));

#using only id

driver.find\_element(By.CSS\_SELECTOR, "#offers").click()

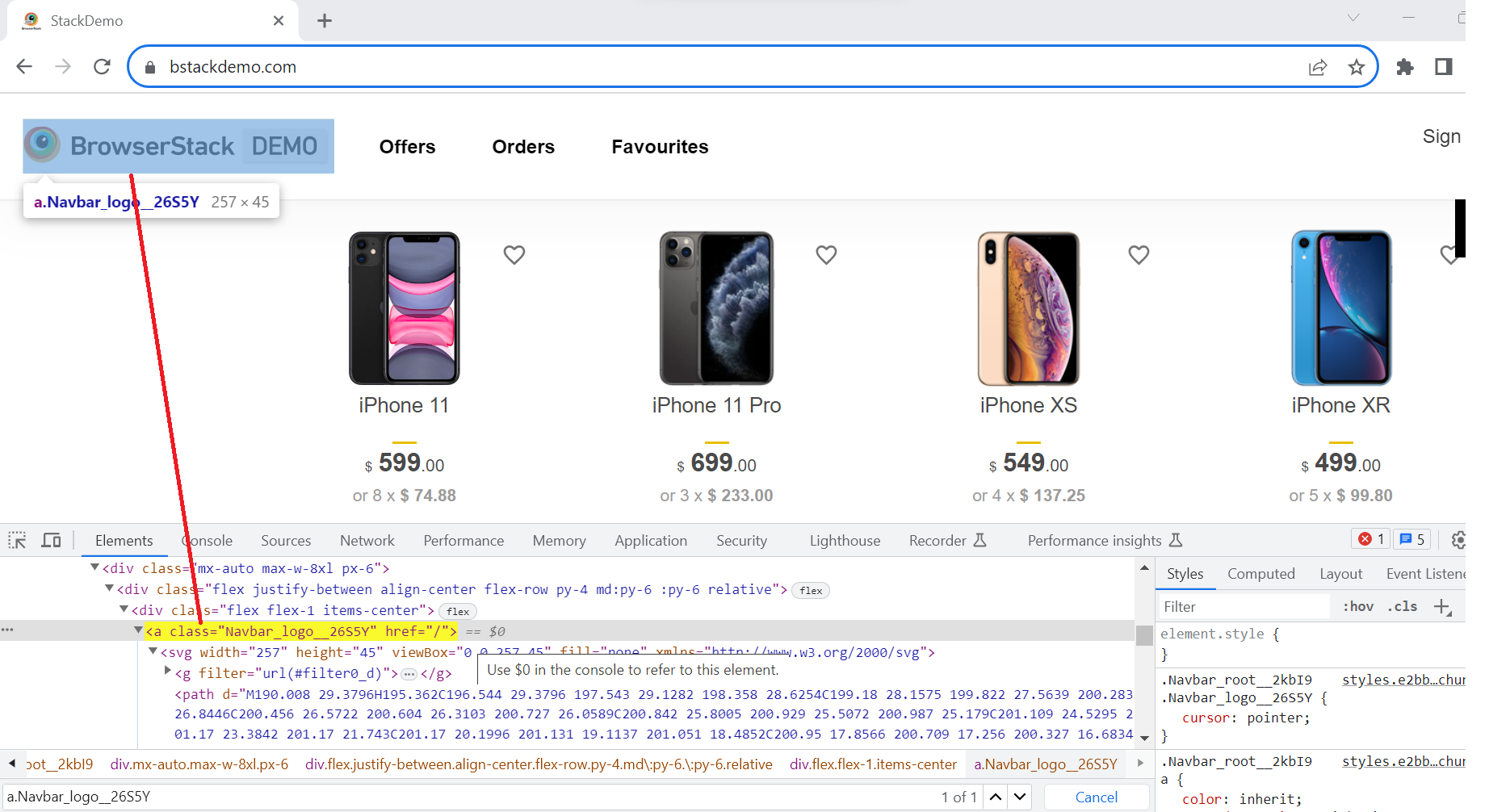
#using with tag name & id

driver.find\_element(By.CSS\_SELECTOR, "a#offers").click()

driver.find\_element(By.CSS\_SELECTOR, "a[id='offers']").click()

**2.Class:**

In CSS, we can use “.” notation to select the “class” attribute of an element.



**Syntax:**

driver.findElement(By.cssSelector(“<tagname>.<class value>”));

driver.findElement(By.cssSelector(“.<class value>”));

driver.findElement(By.cssSelector(“<tagname>[class=’<class value>’]”));

**#using tag name & class**

driver.find\_element(By.CSS\_SELECTOR, "a.Navbar\_logo\_\_26S5Y").click()

driver.find\_element(By.CSS\_SELECTOR, "a[class='Navbar\_logo\_\_26S5Y']").click()

**#using only class name**

driver.find\_element(By.CSS\_SELECTOR, ".Navbar\_logo\_\_26S5Y").click()

#### 3. Attribute:

Apart from “**id**” and “**class**”, other attributes can also be used to locate web elements using CSS selector.



For the WebElement “**Favourites**” tab, tag name is “**a**” and href value is “**/favourites**”.

**Synatax:**

driver.findElement(By.cssSelector(“<tagname>[href=’<href value>’]”));

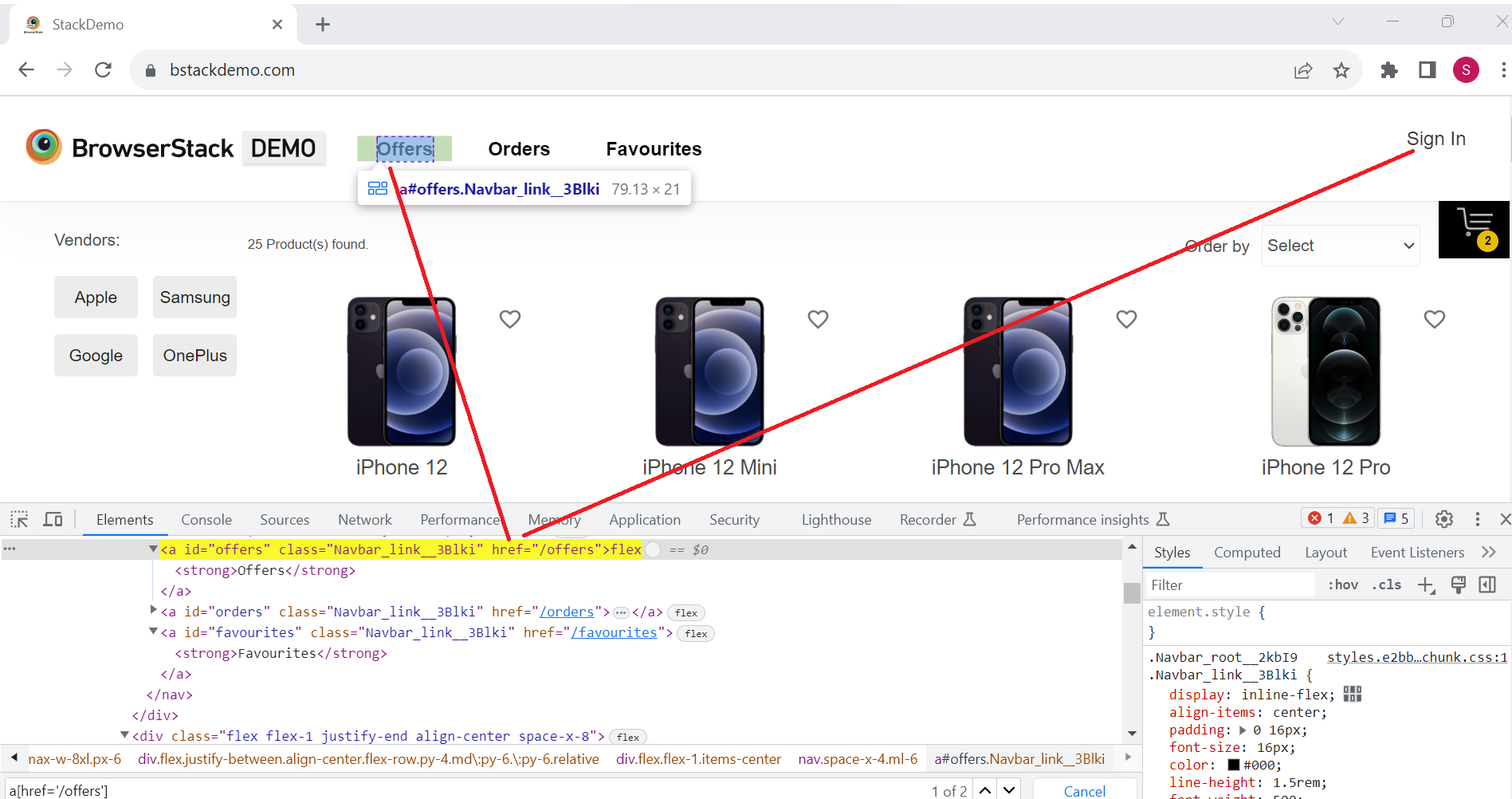
driver.find\_element(By.CSS\_SELECTOR, "a[href='/favourites']").click()

**3.commonattributes:**

From above examples we understood how we can uniquely identify elements using CSS selectors, however sometimes, using only class/ id/ single attribute does not yield a unique locator for a given web element. In such cases, we can combine multiple attributes to fetch a unique locator.

* Id and attribute example:

If we want to locate WebElement “Offers” tab by just one attribute “href”, it gives 2 results which means it is not unique and pointing to 2 web elements on DOM. To make it unique we should also use “id” attribute.



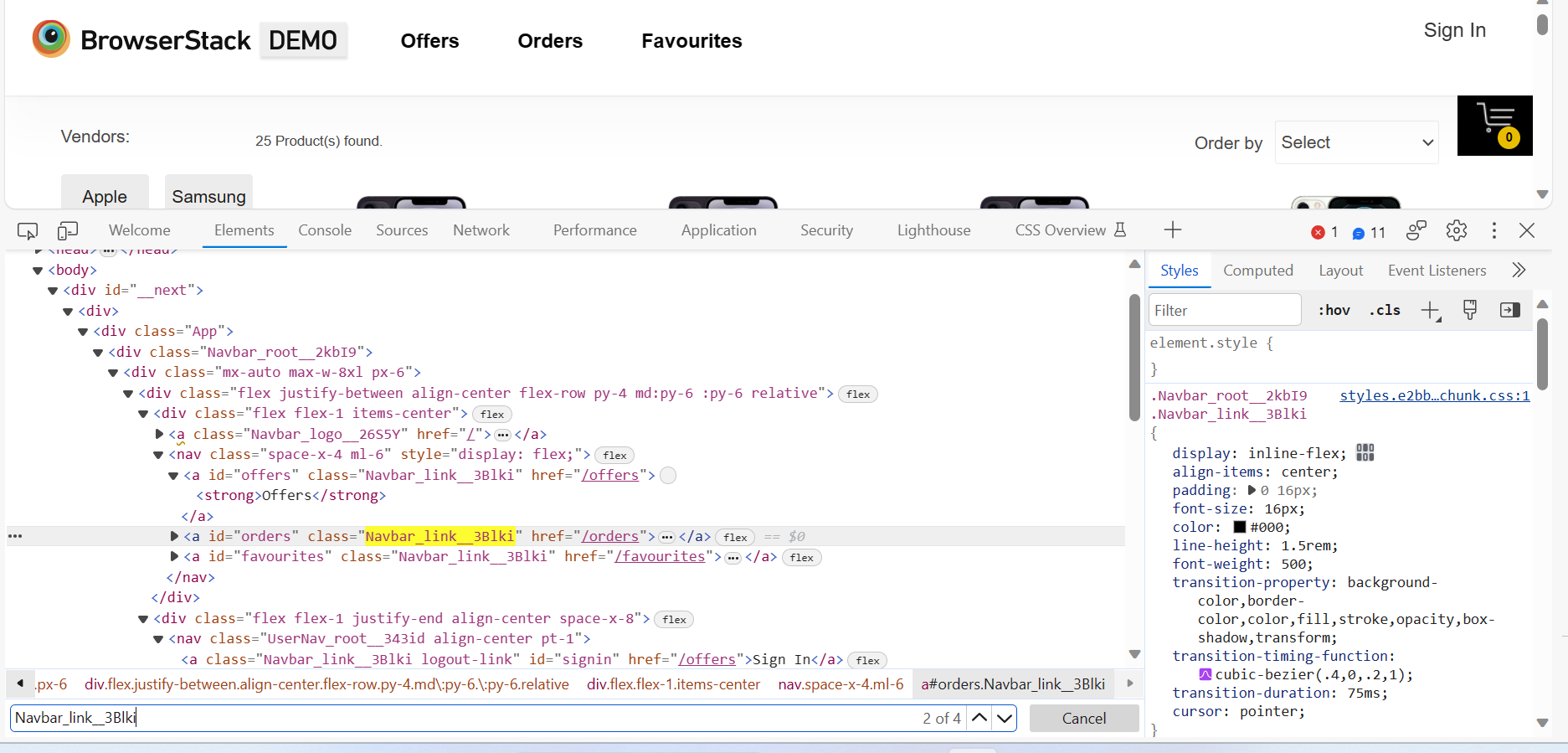
Syntax:

driver.findElement(By.cssSelector(“<tagname>#<id value>[href=’<href value>’]”));

**driver.find\_element(By.CSS\_SELECTOR, "a#offers[href='/offers']").click()**

**Class and attribute Example:**

If we want to locate the WebElement “**Offers**” tab by just class value, it gives 3 results, which means it is not unique and pointing to 3 web elements on DOM. To make it unique we should also use the “**href**” attribute.

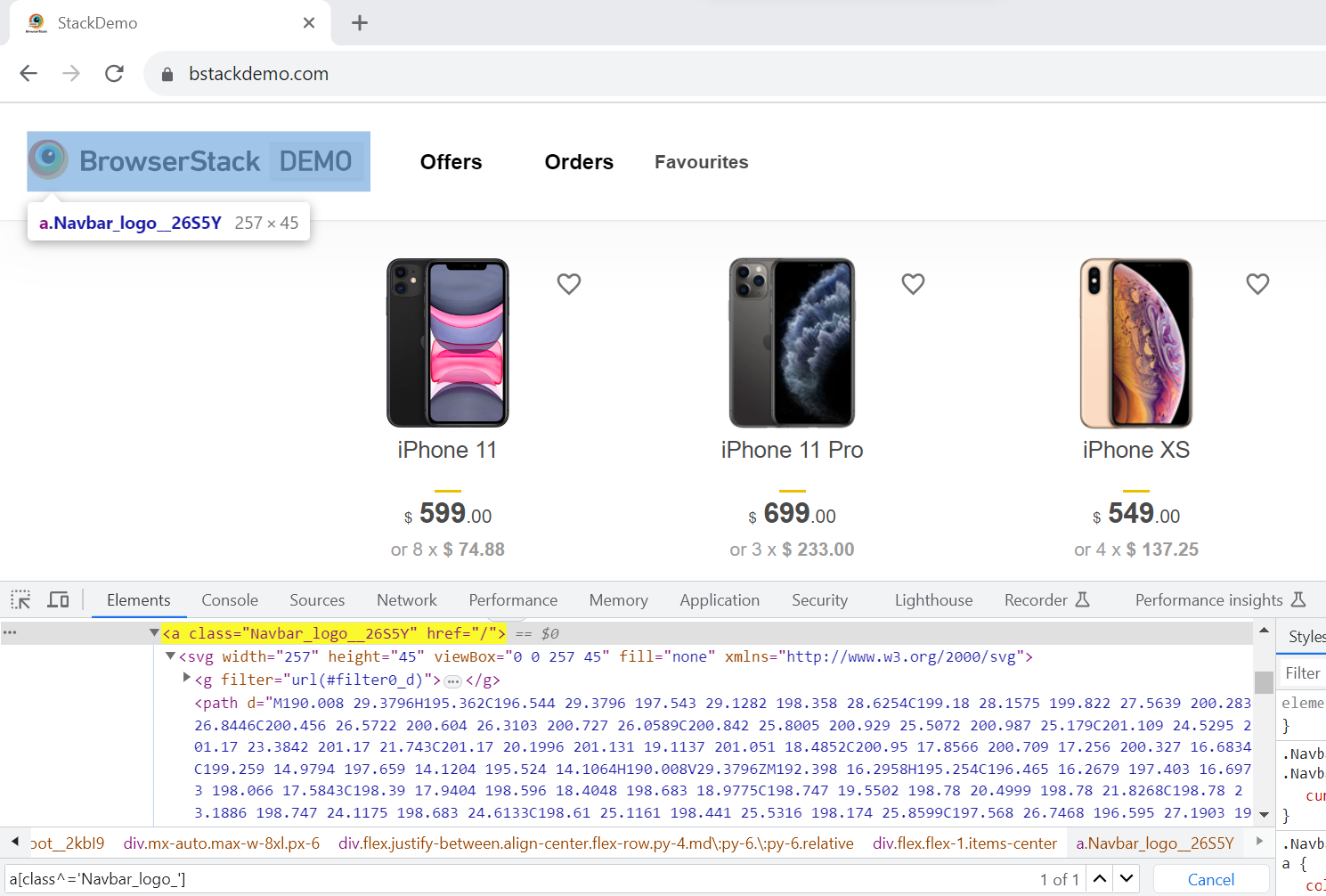


driver.find\_element(By.CSS\_SELECTOR, "a.Navbar\_link\_\_3Blki[href='/orders']").click()

**5.substring:**

CSS Selectors in Selenium allows to match a partial String with the help of various symbols to represent the start, end and contents inside a text. Let us understand all the three ways with example for accessing BrowserStack logo on BStackDemo web application.

* Matching a prefix (Starts with: ^): Locate the web element using the substring that starts with a certain value.



**Syntax:**

driver.findElement(By.cssSelector(“<tagname>[<attribute>^=’prefix of the string’]”));

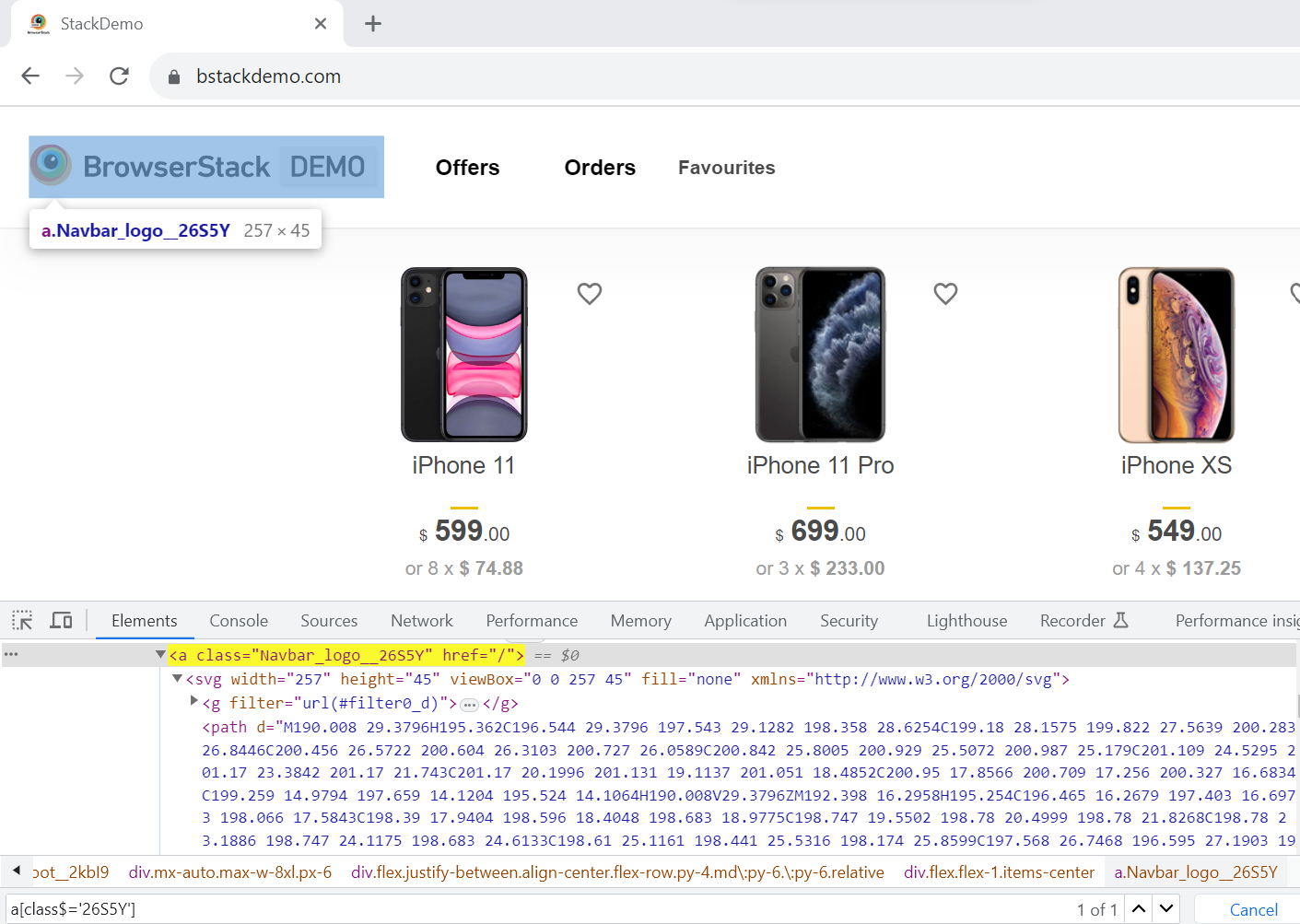
**a[class^=Navbar\_logo]**

**a#offers[class^=Navbar\_link]**

**div[class^=shelf-item]**

* Matching a suffix (Ends with: $): Locate the web element using the substring that ends with a certain value.
* Syntax:

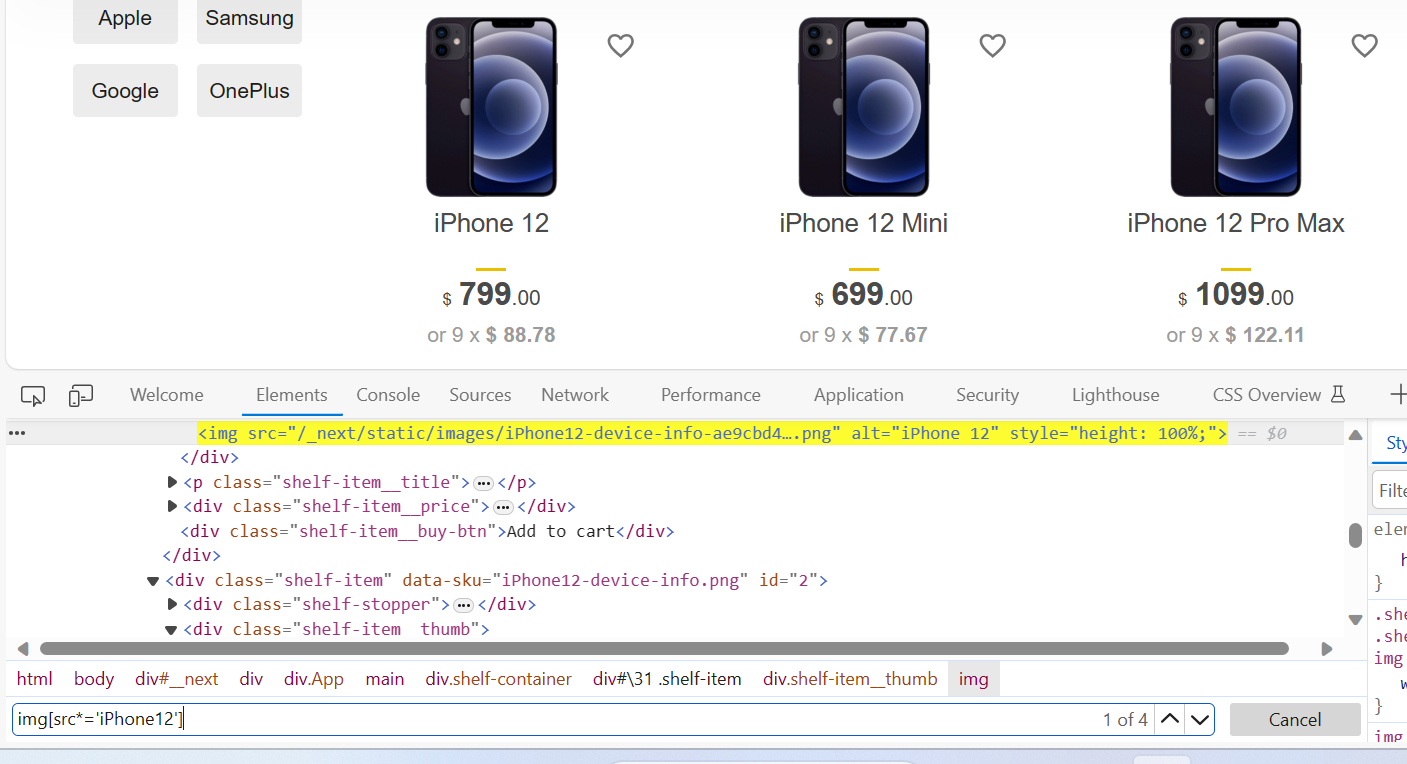
driver.findElement(By.cssSelector(“<tagname>[<attribute>$=’suffix of the string’]”));

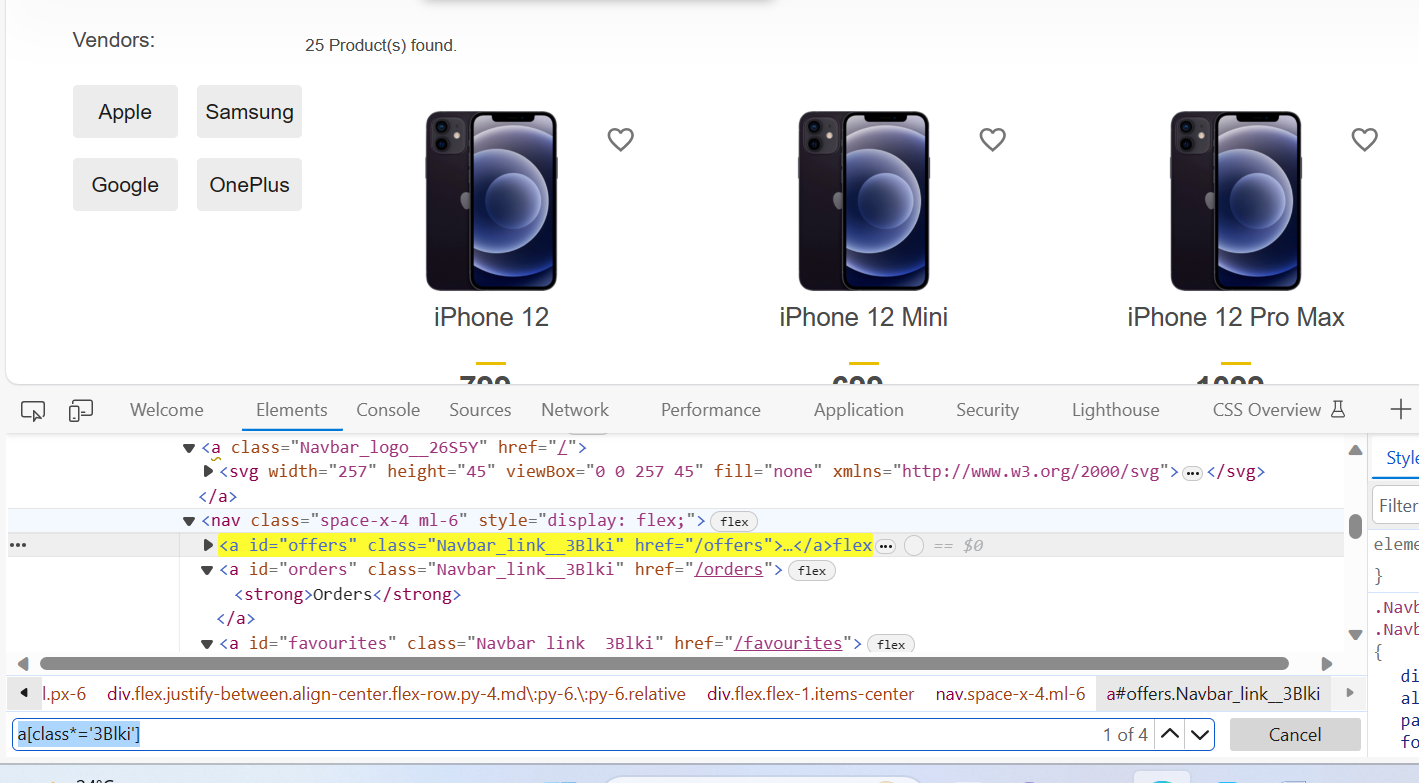


**a[class$='26S5Y']**

****

Matching a substring (contains: \*): Locate the web element by matching the substring.

****

****

**a[class\*='3Blki']**

**span[class\*='mark']**

**XPATH:**

XPath is a Selenium technique to navigate through a page’s HTML structure.

* It enables testers to navigate any document’s XML structure, which can be used on both HTML and XML documents.
* While other [locators in Selenium](https://www.browserstack.com/guide/locators-in-selenium) that search for elements using tags or CSS class names are more straightforward, they may not be sufficient to select all DOM elements of an HTML document.
* XPath provides an option to search for an element within a web page dynamically, thus giving sufficient flexibility to tweak a locator to one’s advantage.

While Selenium has wrappers for most popular programming languages, the selector string remains the same. For instance, one may use the [*.find\_element\_by\_xpath()*method](https://www.browserstack.com/guide/find-element-by-xpath-in-selenium) of the driver class in [Python](https://www.browserstack.com/guide/python-selenium-to-run-web-automation-test), but the locator string that goes as an argument to this method remains the same in all programming languages.

**Types of XPATH in selenium:**

Here is a quick overview of the two types of Selenium XPath:

* **Absolute XPath**: Begins from the root of the HTML document and specifies the complete path to the element. It’s not as flexible and can break if the page structure changes.
* **Relative XPath**: Starts from a specific element and navigates through the DOM hierarchy to locate the desired element. It’s more flexible and resilient to changes in the page structure.

**1.Absolute XPATH:**

**For instance, consider the HTML below**:

<html>

<head>...</head>

<body>

...

<form id="loginForm">

<input name="name" type="text" value="First Name" />

<input name="name" type="text" value="Last Name" />

<input name="email" type="text" value="Business Email" />

<input name="password" type="password" />

<input name="continue" type="submit" value="Sign Me Up" />

</form>

</body>

</html>

The syntax to select the business email field is as follows:

html/body/form/input[3]

This searches for the first form tag in the body of the page and selects the third input field in the form. This format, though simple, is also the most vulnerable to minor changes in the page’s structure. This method is also known as a single slash search.

#### 2. Relative Path:

A relative path, or a double slash search, begins with double slashes. The double slashes signify a break in the absolute path. Here is how to select the same business email field using a relative path.

//form/input[3]

### ****How to handle Dynamic Elements in Selenium using XPath?****

#### 1. Using Attributes

While the example shown above is feasible if only a single form is on the page, one can make the search patterns more robust by using attributes.

//form[@id='loginForm']/input[3]

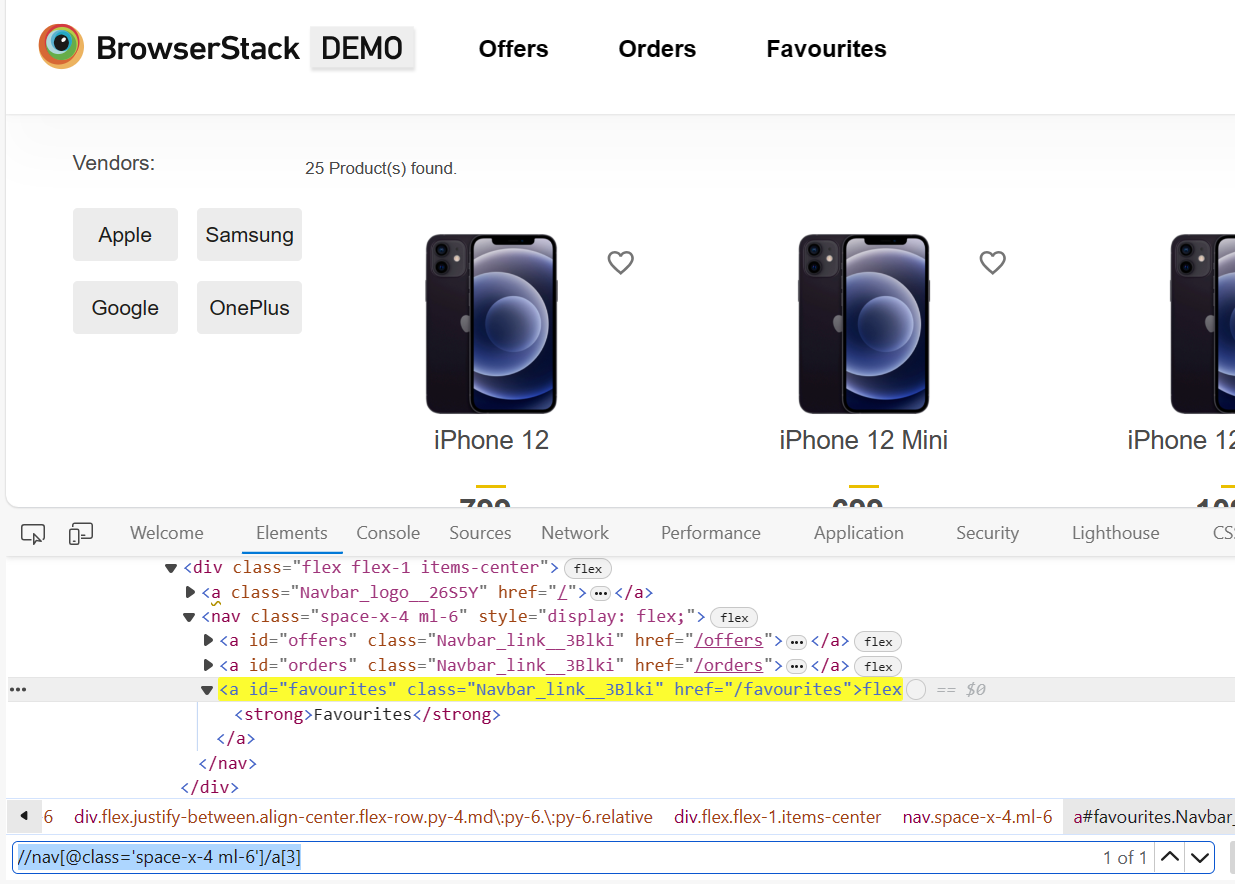
In place of id, one can use any attribute and its corresponding value to locate an element with Selenium.

While this example shows a single attribute, one can also use multiple attributes of the same tag to locate it on the page.

**For instance, to select the Last Name field, one can use the following XPath syntax in Selenium**:

//input[@name='name'][@value='Last Name']

Example:

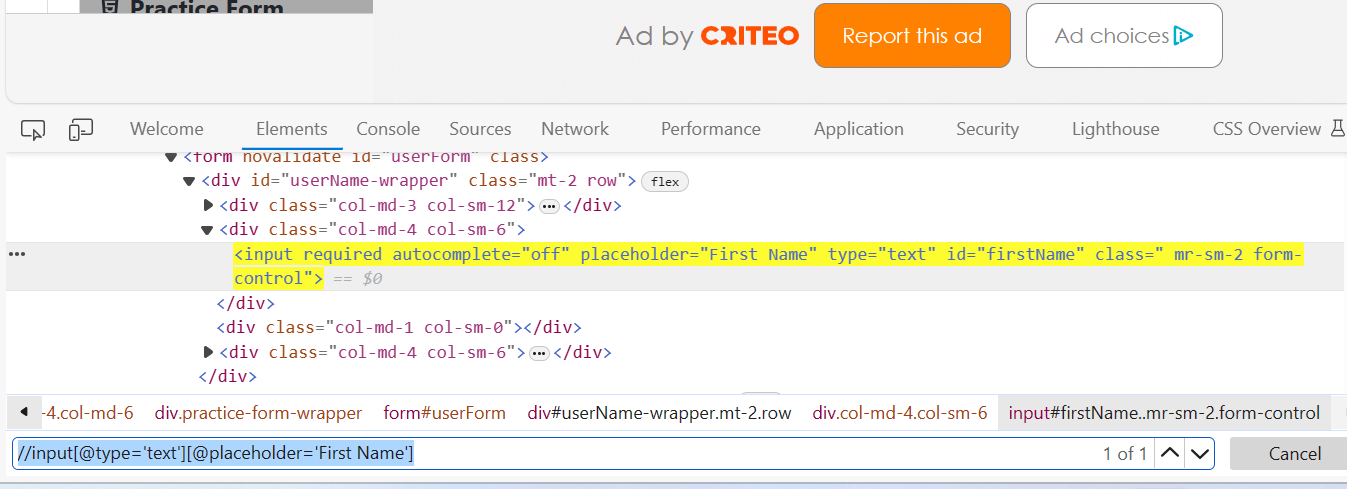


//nav[@class='space-x-4 ml-6']/a[3]

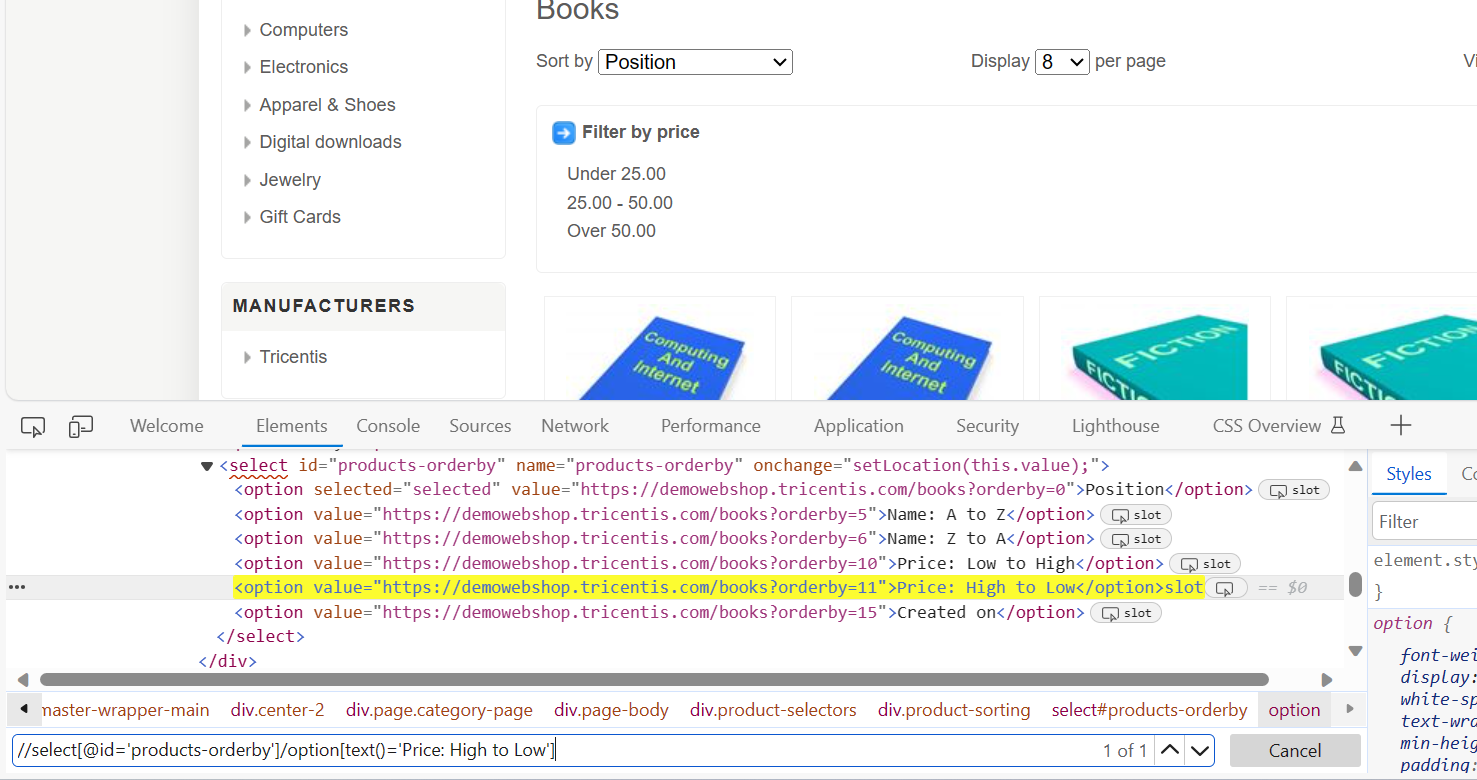
//ul[@id='verticalListContainer']/li[text()='Dapibus ac facilisis in']



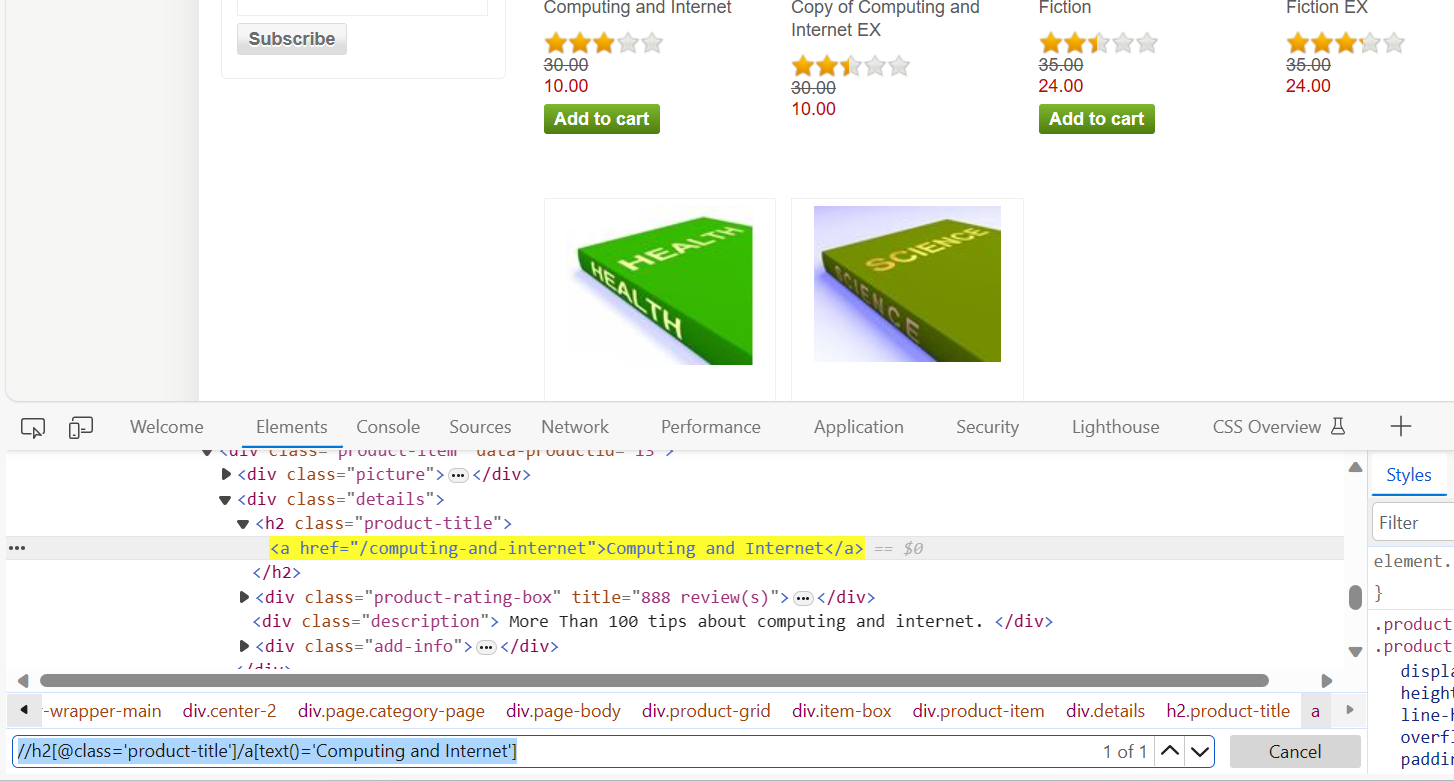
//div[@class='rt-tr -odd'][@role='row']



//input[@type='text'][@placeholder='First Name']



//select[@id='products-orderby']/option[text()='Price: High to Low']



//h2[@class='product-title']/a[text()='Computing and Internet']

//div[@class='product-viewmode']/select[@name='products-viewmode']

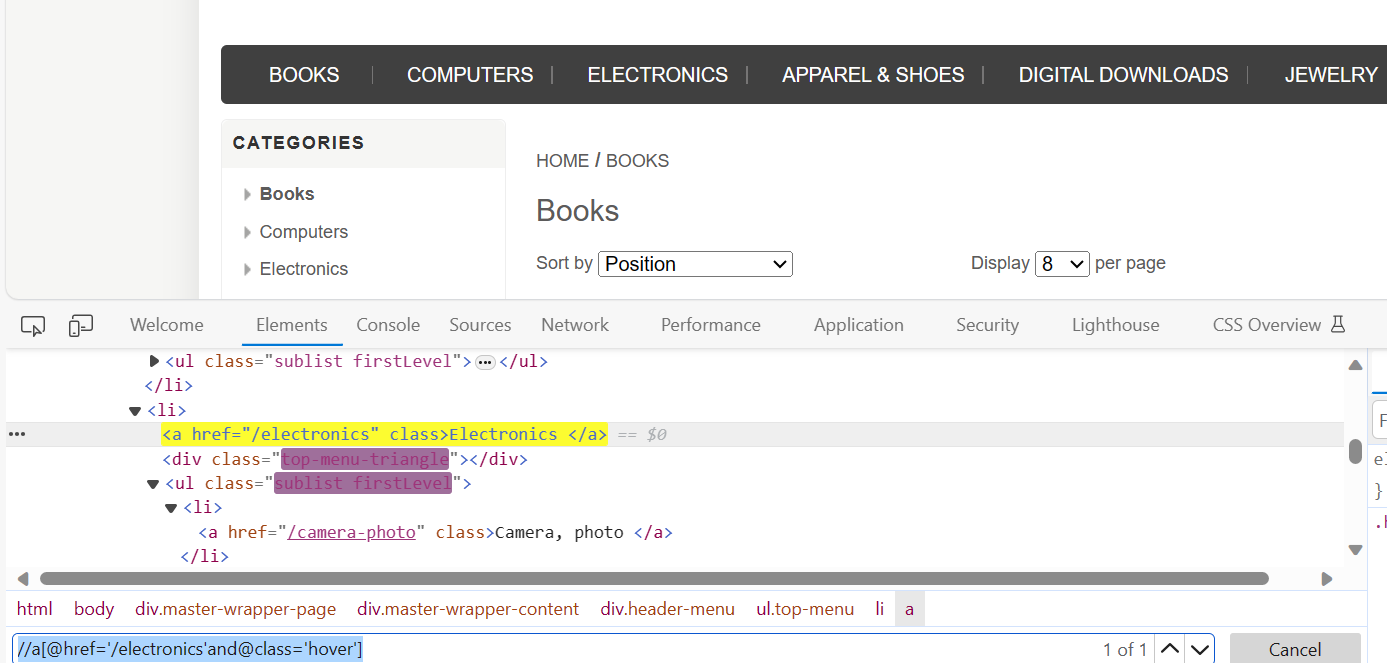
//select[@name='products-viewmode']/option[text()='List']

//div/h2/a[text()='Computing and Internet']

#### ****2. Logical Operators in Selections:****

While attributes may be sufficient to locate elements in most cases, testers may also need to use logical operators. For instance, if the HTML structure has name or id attributes populated by the value “name”, one may use the following syntax to select them.

//input[@id='name' or @name='name']



//a[@href='/electronics'and@class='hover']

//a[@href='/camera-photo' and @class='hover']

//li/a[@href='/desktops' and @class='hover']

’

//ul/li/input[@type='radio' and @name='pollanswers-1']

//ul/li/label[@for='pollanswers-1' and text()='Excellent']

//ul/li/label[@for='pollanswers-1'][text()='Excellent']

#### 3. Using Text:

One may search for an element using the text that it contains too. For instance, to select a link that says “Click Me”, one can use the following search:

//a[text()='Click Me']

This snippet searches for any hyperlink containing the text “Click Me”. Replace the tag with a wildcard *\** to search for any element that contains the text “Click Me”.

//\*[text()='Click Me']

//\*[text()='Register']

//\*[@class='ico-cart']/span[text()='Shopping cart']

**Contains:**

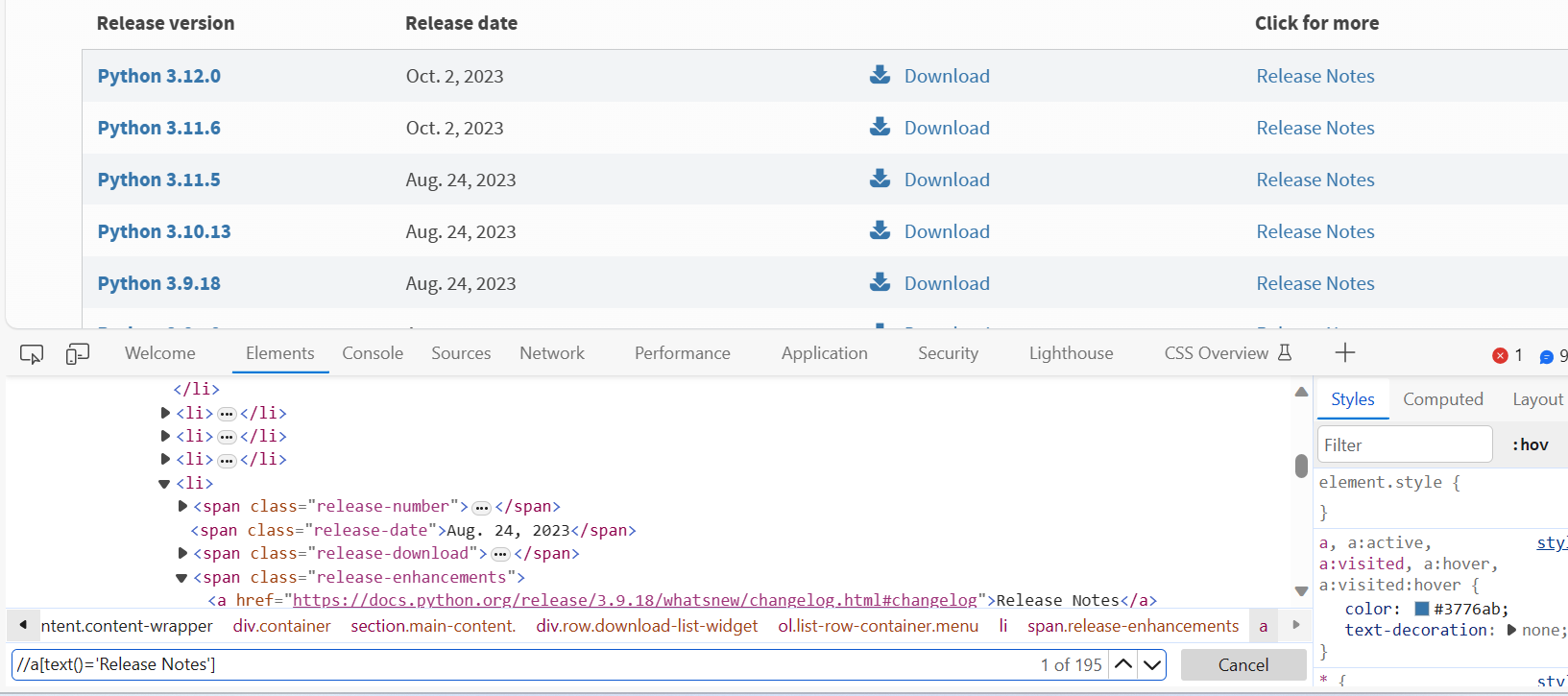
If the text or attribute value of the element has leading or/and trailing white spaces, we need to use “contains”

//HTMLTAG[contains(@attribute, “attribute\_value”)]

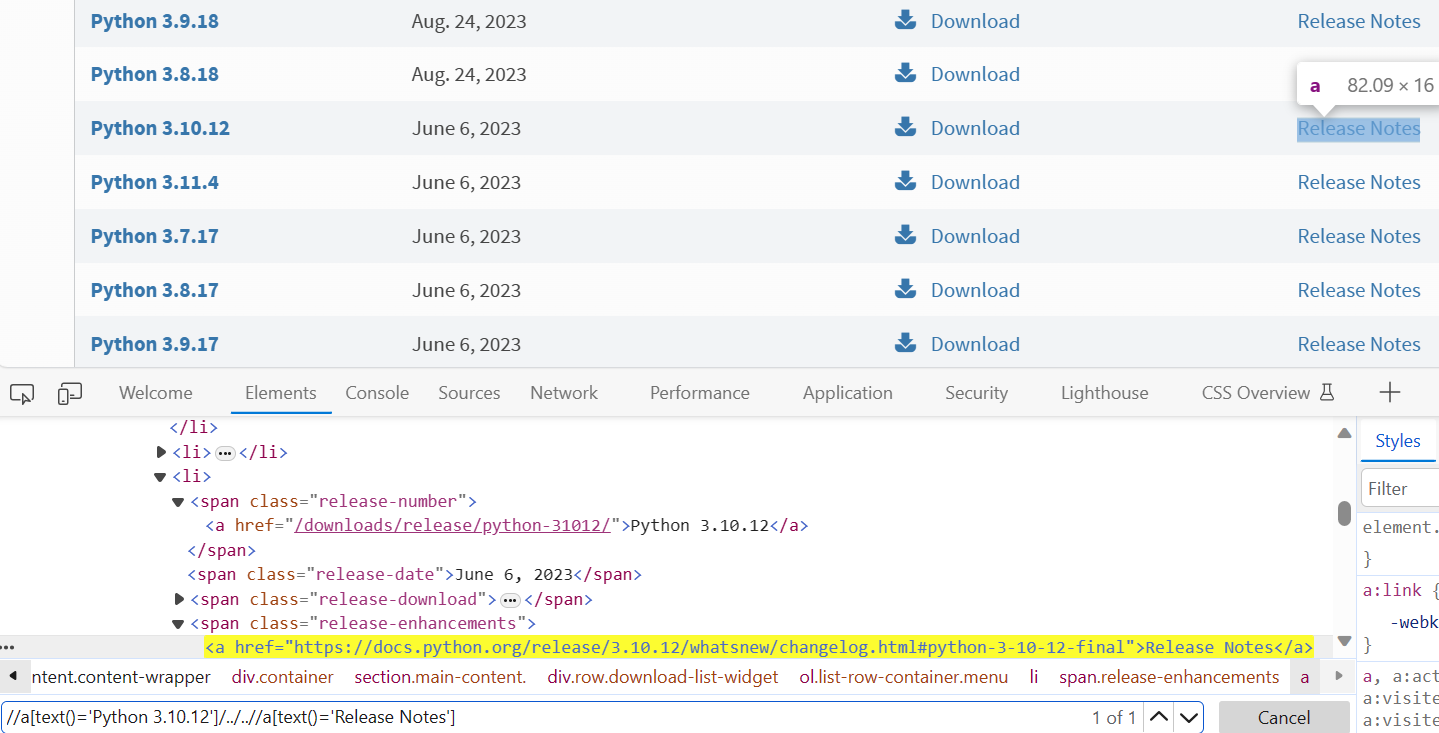
//HTMLTAG[contains(text(), “actual\_text”)]

**Dependent Independent technique:**

Example1:



Solution:

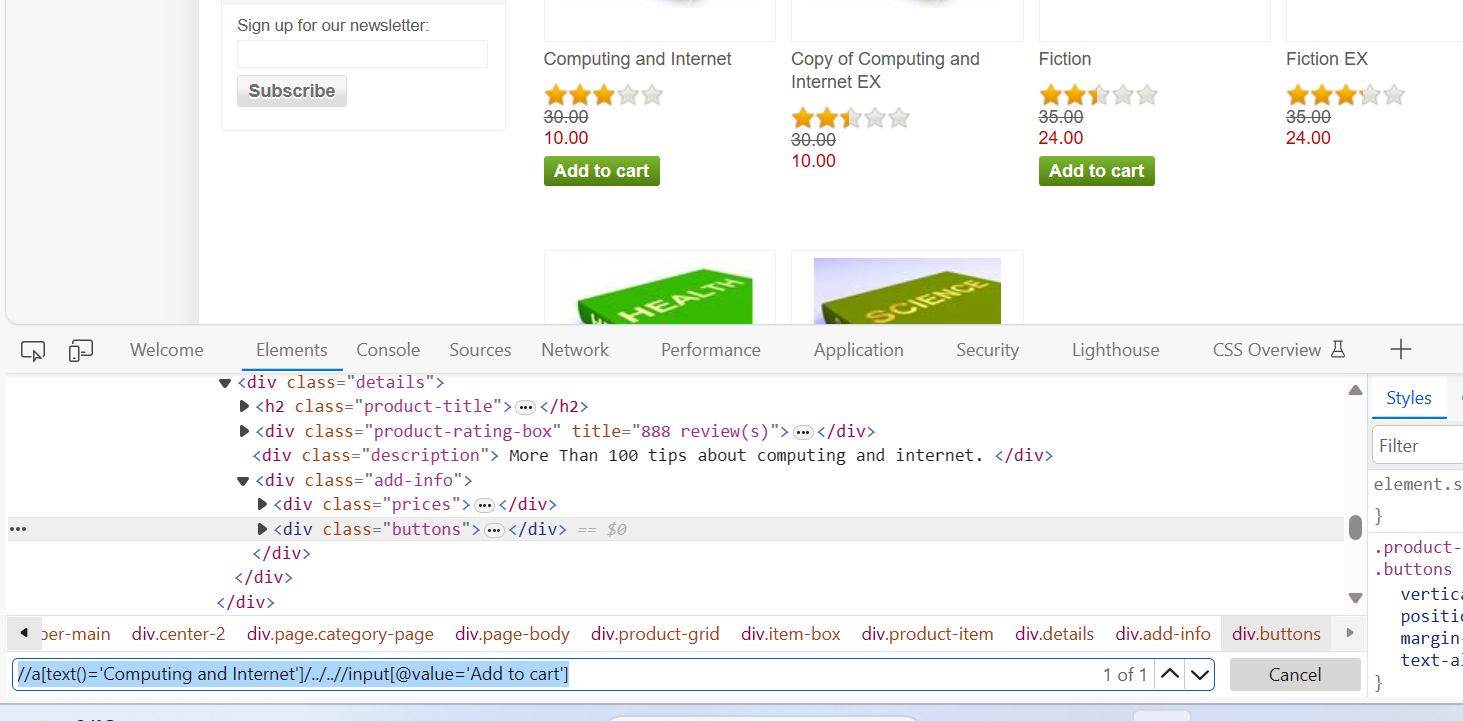


//a[text()='Python 3.10.12']/../..//a[text()='Release Notes']

All releases notes elements are dependent items which are duplicates, so every release notes are dependent on python versions. So we can start from taking xpath of python version and can traverse upto release notes using

/.. and ending we need to write xpath for Release notes

Example2:



//a[text()='Computing and Internet']/../..//input[@value='Add to cart']

Conftest:

* if you give scope='function' in conftest it will run the fixture with every method in a class and before  
  all normal functions also  
  means it will open seperate browser for each normal functions (if you created test file using only normal functions)  
  &it will open a seperate browser for each method in a class (if you created test file with test methods inside a class)
* if you give sope='class' in conftest it will run the fixture with class.  
  means it will run the conftest once with one class and all methods in the class  
  so all the methods in the class executed in the same browse

Example conftest:

**Conftest.py:**

from pytest import fixture  
from selenium import webdriver  
@fixture(scope='class')  
def setup():  
 driver = webdriver.Chrome()  
 driver.get(r"https://www.ajio.com")  
 driver.implicitly\_wait(5)  
 driver.maximize\_window()  
 yield driver  
 driver.close()

Test\_file.py

from ajio\_submission.page\_home.pages.home\_page import menu\_header\_validation  
from ajio\_submission.page\_home.pages.items\_list\_page import home\_items\_list\_page  
from time import sleep  
  
@pytest.mark.usefixtures('setup')  
class Test\_home\_page\_header():  
  
  
 def test\_menu\_list(self, setup):  
 menu\_obj1 = menu\_header\_validation(setup,  
 r"C:\development\pom\_frame\_work\ajio\_submission\page\_home\utilities\test\_data\_fk.xlsx",  
 'locators')  
 menu\_obj1.mouse\_hovering\_to\_men\_menu()  
 menu\_obj1.mouse\_hovering\_to\_woomen\_menu()  
 menu\_obj1.mouse\_hoverig\_to\_kids\_menu()  
 menu\_obj1.mouse\_hoverig\_to\_indie\_menu()  
 menu\_obj1.mouse\_hoverig\_to\_home\_kitechen\_menu()  
  
 *#* def test\_items\_list\_heading\_validation(self, setup):  
 heading\_obj1 = home\_items\_list\_page(setup,  
 r"C:\development\pom\_frame\_work\ajio\_submission\page\_home\utilities\test\_data\_fk.xlsx",  
 'locators')  
 heading\_obj1.navigating\_to\_men\_menu\_and\_clicking\_backpacks()  
 heading\_obj1.items\_list\_page\_header\_validating()

**Using conftest fixture in a python class:**

From pytest import mark

@mark.usefixtures(‘coftest\_file\_name’) #here conftest file name itself acts as a driver

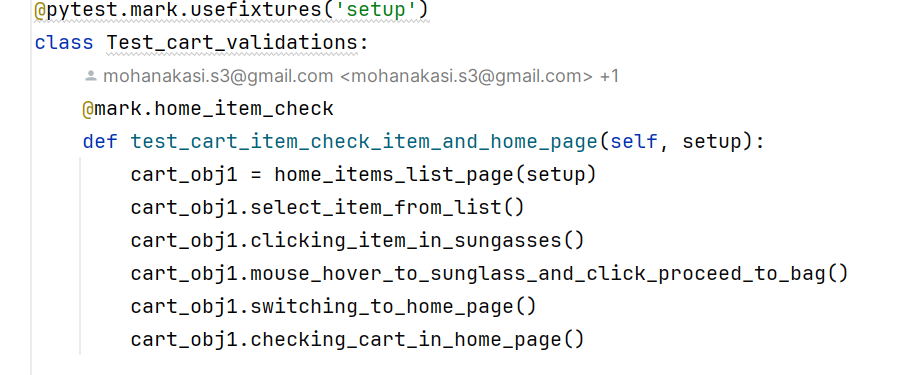
Class Test\_class1:

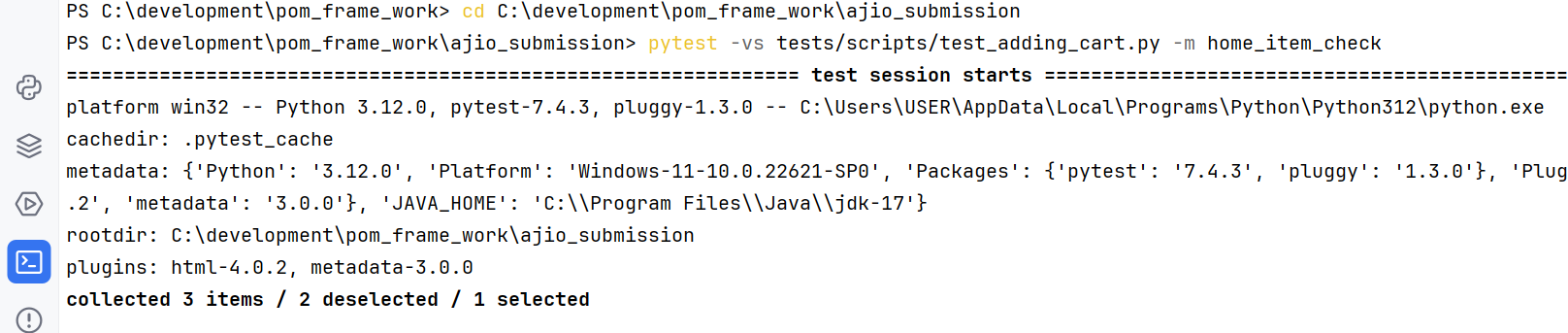
Pass

pytest grouping tests:

from pytest import mark

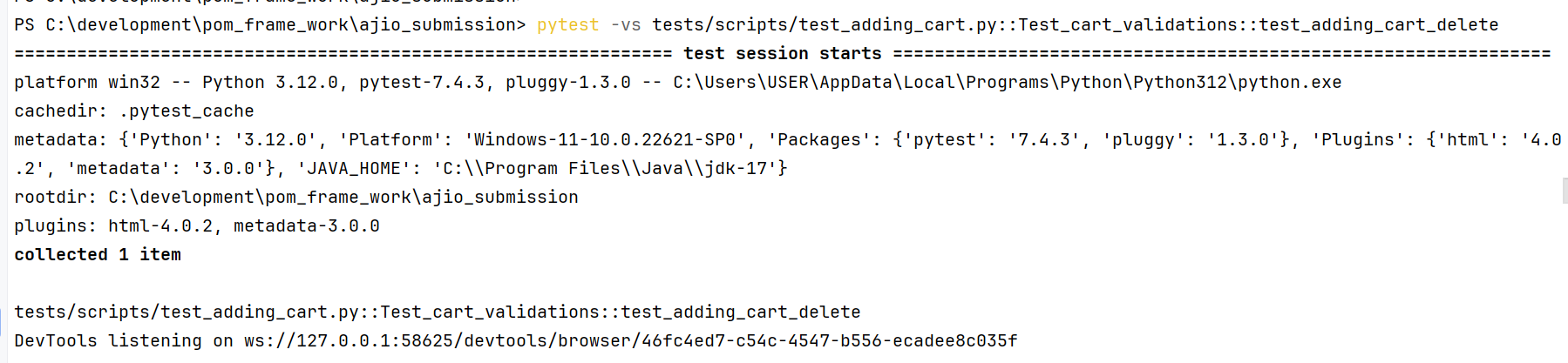
@mark.mark\_name





Executing a particular method in a test file:

pytest file\_name.py::class\_name::test\_method\_name

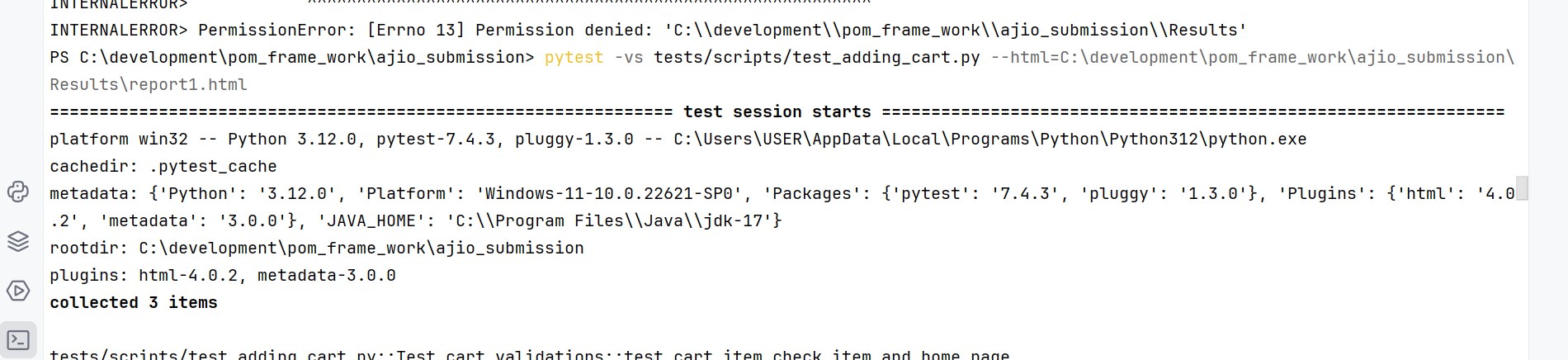


Generating html reports:

First you need to install html pluggin to generate reports

pip install pytest-html

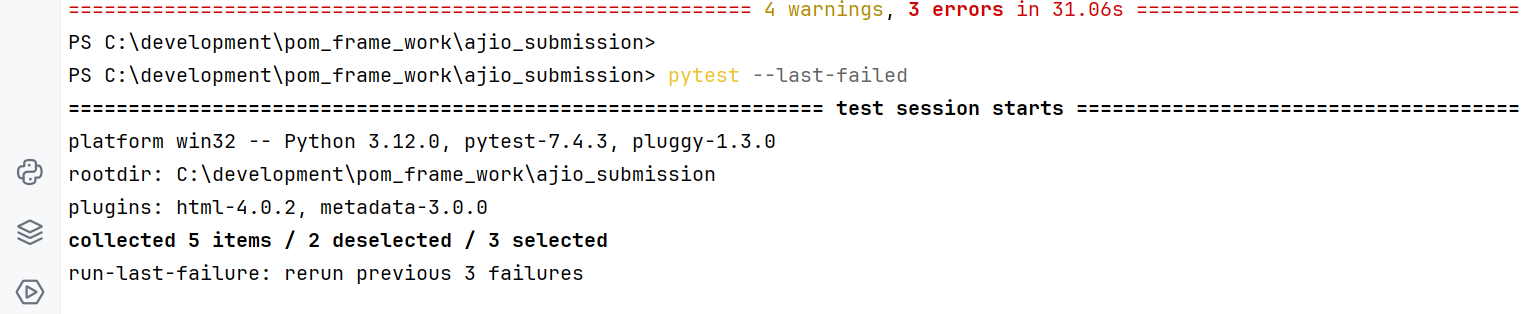
pytest file\_name.py --html=file\_name.html



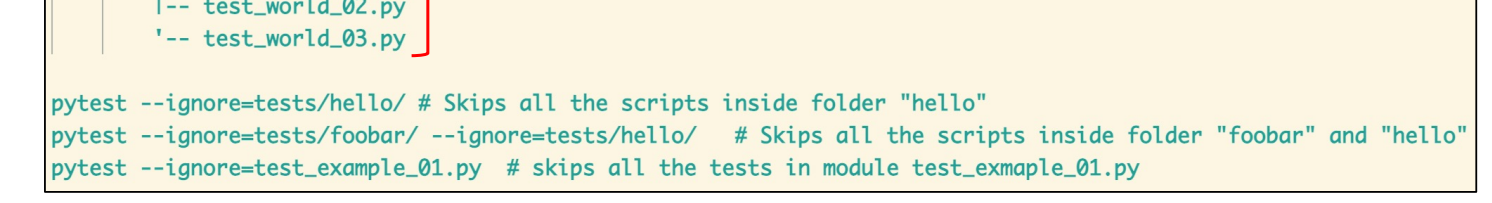


**Running only failed scripts:**

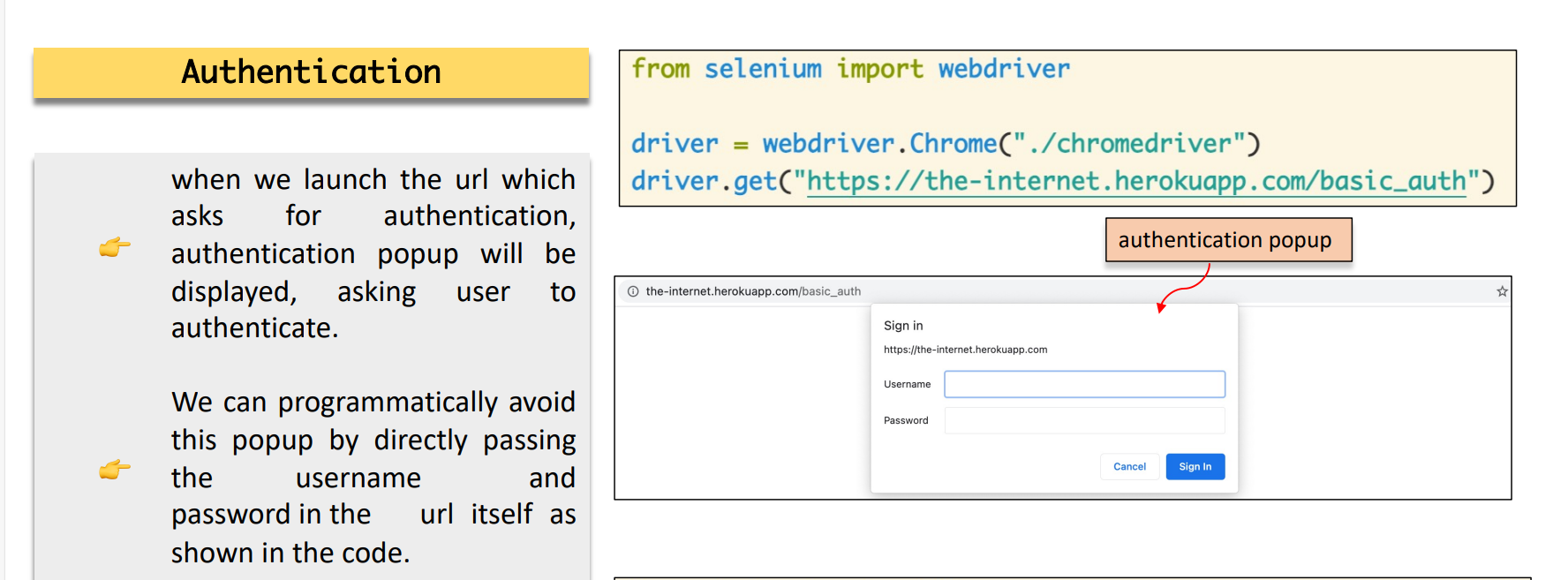
pytest --last-failed

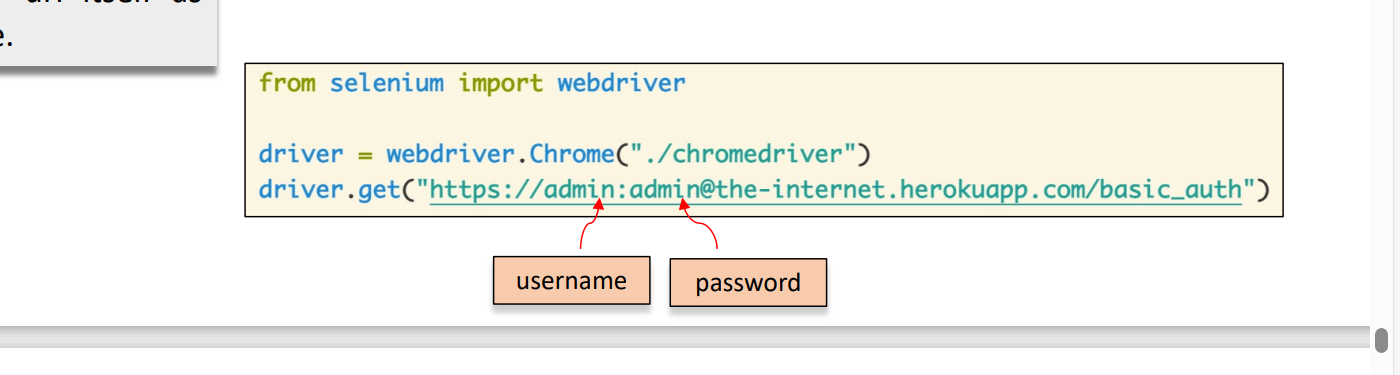


**Ignoring sritps:**



**Handling authentication pop-ups:**





Example url:

<http://the-internet.herokuapp.com/basic_auth>