Selenium Automation

Selenium is a popular tool used in software development and testing. It is specifically used for automating web browsers. In simple terms, it allows developers to write code that can interact with web browsers like Chrome, Firefox, or Safari.

With Selenium, you can perform various tasks that a human user would do on a website. For example, you can open a webpage, fill out forms, click buttons, navigate through different pages, and extract data from web pages.

Let's say you want to test a login functionality on a website. Instead of manually opening the browser, entering the username and password, and clicking the login button every time, you can use Selenium to automate this process. You can write code that instructs the browser to open the login page, enter the username and password, and click the login button. Selenium allows you to simulate these actions programmatically.

step : 1 Pip install selenium

step :2

```
Certainly! Here are the links to download the WebDriver executables for th
e most commonly used web browsers:
```

1. ChromeDriver (for Google Chrome):

   - Download: https://sites.google.com/a/chromium.org/chromedriver/downloads (https://sites.google.com/a/chromium.org/chromedriver/downloads)
2. GeckoDriver (for Mozilla Firefox):

   - Download: https://github.com/mozilla/geckodriver/releases (https://github.com/mozilla/geckodriver/releases)
3. Microsoft WebDriver (for Microsoft Edge):

   - Download: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/ (https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/)
4. SafariDriver (for Safari on macOS):

   - Pre-installed on macOS. You can enable it from Safari preferences.

Make sure to download the appropriate WebDriver executable for the version of the web browser you have installed on your system. Extract the executable file and provide its path when creating an instance of the corresponding WebDriver class.

Additionally, if you are using a different browser or need to download WebDriver for a specific version, you can usually find the appropriate WebDriver executable by searching for the browser name followed by "WebDriver" on a search engine. For example, "Opera WebDriver" or "Internet Explorer

Simple Test

In [8]:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

# Instantiate the Chrome driver
driver = webdriver.Chrome()

# Open the desired URL
driver.get("http://orangehrm.qedgetech.com/symfony/web/index.php/auth/validateCrede
```

In [9]:

```
# Find an element by name and send keys
driver.find_element(By.NAME, "txtUsername").send_keys('Admin')
driver.find_element(By.ID ,"txtPassword").send_keys('admin123')
```

In [10]:

```
# click
driver.find_element(By.ID,"btnLogin").click()
```

In [11]:

```
# Get the actual title

act_title=driver.title
exp_title='OrangeHRM'
```

In [12]:

```
# Compare the actual and expected titles
if act_title==exp_title:
    print('test passed')
else:
    print('print failed')

driver.close()
```

```
test passed
```

Explanation

the Selenium library is imported, and the Chrome WebDriver is instantiated to automate the Chrome browser. Then, a specific URL is opened using the get() method of the driver object. After that, the code enters the username and password on the login page using send_keys() method of the driver object to locate the input fields by their name and ID.

The act_title variable is assigned the title of the current web page, which is retrieved using the title attribute of the driver object. The exp_title variable is assigned the expected title value.

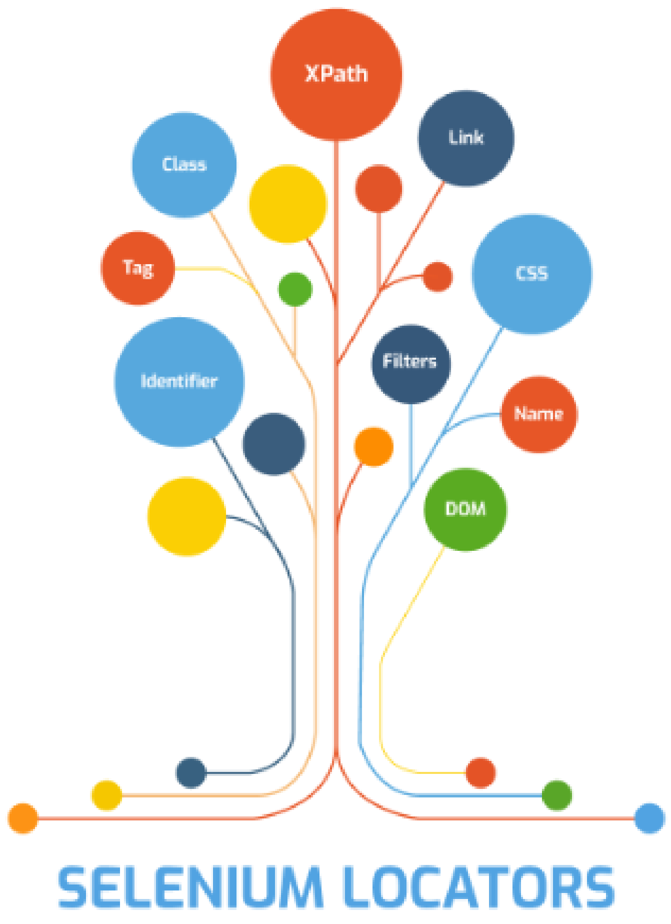
In the focal cell, an if statement is used to compare the actual title (act_title) with the expected title (exp_title). If they are equal, the code prints "test passed". Otherwise, it prints "test failed".

Finally, the driver.close() method is called to close the browser session.

In this particular case, the output of the focal cell is "test passed", indicating that the actual title matches the expected title.

What are Locators?

In Selenium, locators are used to identify and locate web elements on a web page. Selenium provides several locator strategies to find elements based on different attributes and properties. Here are some commonly used locator strategies in Selenium:



In Selenium, locators are used to identify and locate web elements on a web page. Selenium provides several locator strategies to find elements based on different attributes and properties. Here are some commonly used locator strategies in Selenium:

1. **ID**:

   - Example: `driver.find_element(By.ID, 'element_id')`
   - Locates an element by its unique ID attribute.
2. **Name**:

   - Example: `driver.find_element(By.NAME, 'element_name')`
   - Locates an element by its name attribute.
3. **Class Name**:

   - Example: `driver.find_element(By.CLASS_NAME, 'element_class')`
   - Locates an element by its class attribute.
4. **Tag Name**:

   - Example: `driver.find_element(By.TAG_NAME, 'element_tag')`
   - Locates elements by their HTML tag name.
5. **Link Text**:

   - Example: `driver.find_element(By.LINK_TEXT, 'link_text')`
   - Locates a link element by the exact text displayed.
6. **Partial Link Text**:

   - Example: `driver.find_element(By.PARTIAL_LINK_TEXT, 'partial_link_text')`

- Locates a link element by a partial match of the displayed text.

7. **CSS Selector**:
   - Example: `driver.find_element(By.CSS_SELECTOR, 'css_selector')`
   - Locates an element using a CSS selector.
8. **XPath**:
   - Example: `driver.find_element(By.XPATH, 'xpath_expression')`
   - Locates an element using an XPath expression.

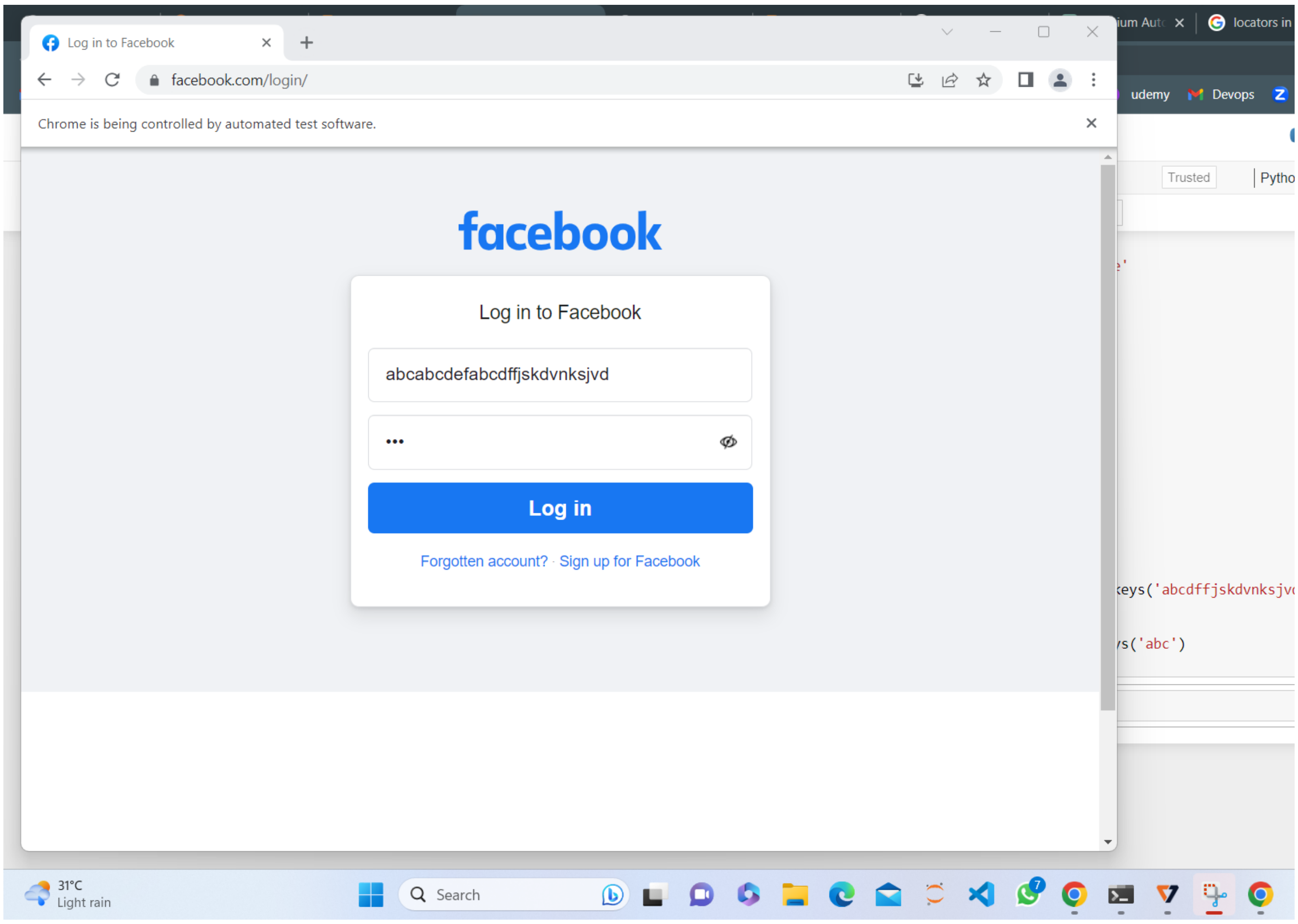
The **driver.find_element()** method is used to locate a single element, while **driver.find_elements()** is used to locate multiple elements that match the specified locator strategy.

Note: The locators can be used with both **find_element()** and **find_elements()** methods, and you need to import the **By** class from **selenium. webdriver. common.by** to use these locators.

Remember that the choice of locator strategy depends on the specific attributes and structure of the web page you are working with. Inspecting the HTML structure of the page can help identify suitable locators for the elements you want to interact with.

In [6]:

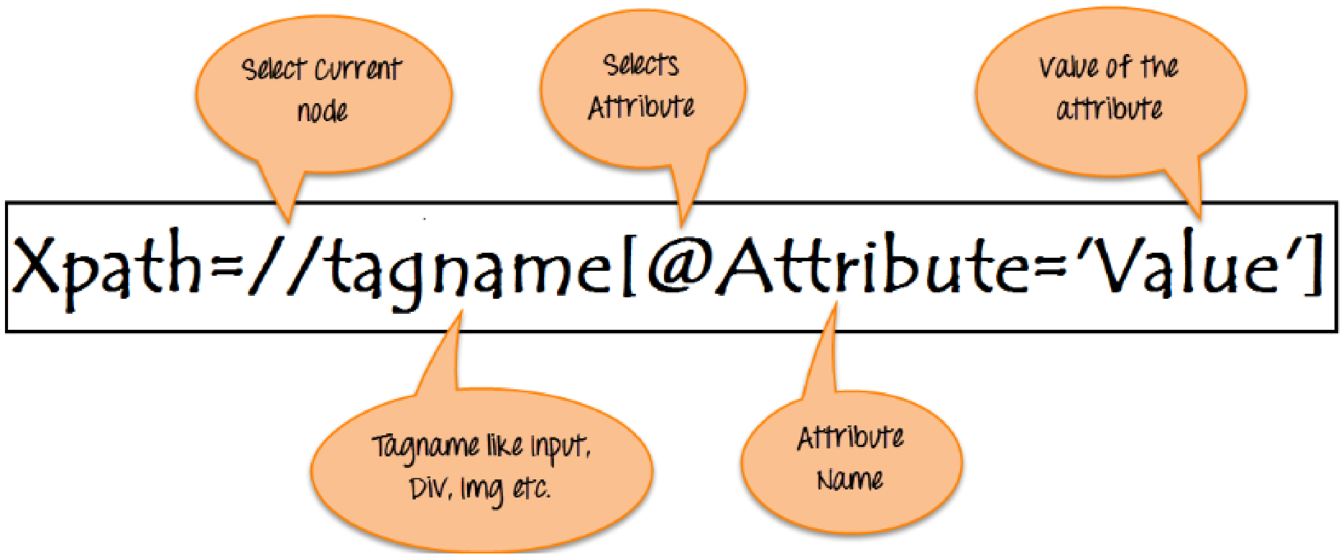
```python
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

# Specify the path to the ChromeDriver executable
chromedriver_path = 'C:\\Users\\user\\PycharmProjects\\selenium\\day 1 selenium\\ch

# Create a Service object with the ChromeDriver executable
service_object = Service(chromedriver_path)

# Create a new instance of the Chrome driver, using the Service object
driver = webdriver.Chrome(service=service_object)

# Open the Facebook login page
driver.get('https://www.facebook.com/login/')

# Find element using Tag and ID, and send keys
driver.find_element(By.CSS_SELECTOR, 'input#email').send_keys('abc')

# Find element using Tag and ClassName, and send keys
driver.find_element(By.CSS_SELECTOR, 'input.inputtext').send_keys('abcdef')

# Find element using Tag and Attribute, and send keys
driver.find_element(By.CSS_SELECTOR, 'input[placeholder="Email address or phone num

# Find element using Tag, ClassName, and Attribute, and send keys
driver.find_element(By.CSS_SELECTOR, 'input.inputtext[autocomplete="current-passwor
```


Log in to Facebook
facebook.com/login/
Chrome is being controlled by automated test software.
facebook
Log in to Facebook
abcabcdefabcdffjskdvnksjvd
•••
Log in
Forgotten account? · Sign up for Facebook

```
keys('abcdffjskdvnksjv
ys('abc')
```

2. What are Xpath?

XPath is a powerful locator strategy that allows you to navigate and select elements in an XML or HTML document. Selenium supports XPath as a locator strategy to find web elements. XPath expressions can be used to locate elements based on their attributes, text content, hierarchical relationships, and more.



XPath is a powerful locator strategy that allows you to navigate and select elements in an XML or HTML document. Selenium supports XPath as a locator strategy to find web elements. XPath expressions can be used to locate elements based on their attributes, text content, hierarchical relationships, and more.

Here are some examples of XPath expressions:

1. **Locate an element with a specific ID**:
   - XPath: //*[@id="element_id"]
   - Example: driver.find_element(By.XPATH, '//*[@id="element_id"]')
2. **Locate an element by its attribute value**:
   - XPath: //tag_name[@attribute_name="attribute_value"]
   - Example: driver.find_element(By.XPATH, '//input[@name="username"]')
3. **Locate an element by its text content**:
   - XPath: //tag_name[text()="desired_text"]
   - Example: driver.find_element(By.XPATH, '//a[text()="Click Here"]')
4. **Locate an element based on its relationship to other elements**:
   - XPath: //parent_tag/child_tag`
   - Example: driver.find_element(By.XPATH, '//div[@class="parent"]/span')
5. **Locate an element using multiple conditions:**
   - XPath: //tag_name[@attribute1="value1" and @attribute2="value2"]
   - Example: driver.find_element(By.XPATH, '//input[@type="text" and @name="username"]')

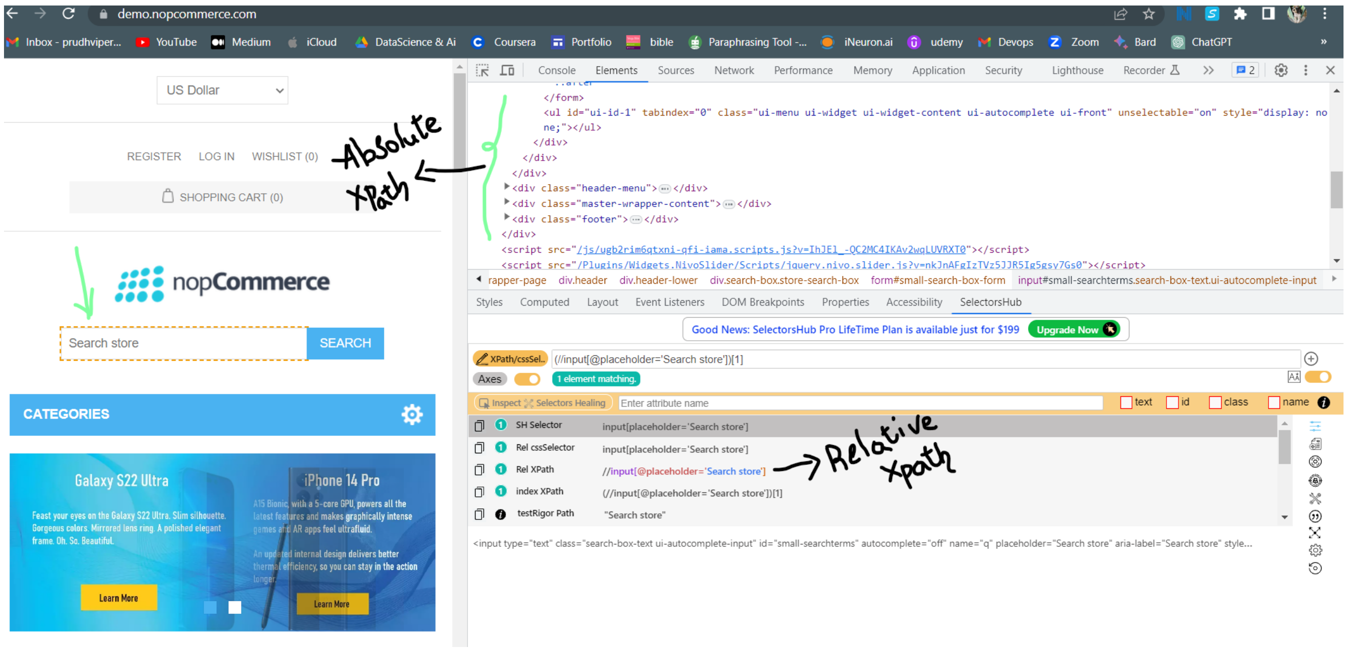
XPath expressions provide great flexibility in locating elements, allowing you to specify precise criteria to match the desired elements. However, it's important to note that complex XPath expressions can be less readable and more prone to breaking if the structure of the web page changes.

When using XPath, you can pass the XPath expression as the locator argument to the `driver.find_element()` or `driver.find_elements()` method, along with the `By.XPATH` constant.

XPath expressions can be constructed using various syntax rules and functions. It's recommended to familiarize yourself with XPath syntax and practices to make effective use of XPath locators in Selenium.

In [14]:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service

# Specify the path to the ChromeDriver executable
chromedriver_path = 'C:\\Users\\user\\PycharmProjects\\selenium\\day 1 selenium\\ch

# Create a Service object with the ChromeDriver executable
service_object = Service(chromedriver_path)

# Create a new instance of the Chrome driver, using the Service object
driver = webdriver.Chrome(service=service_object)

# Open the website
driver.get('https://demo.nopcommerce.com/')

# ABSOLUTE XPATH
driver.find_element(By.XPATH, '/html[1]/body[1]/div[6]/div[1]/div[2]/div[2]/form[1]
driver.find_element(By.XPATH, '/html[1]/body[1]/div[6]/div[1]/div[2]/div[2]/form[1]
```



```html
        </form>
        <ul id="ui-id-1" tabindex="0" class="ui-menu ui-widget ui-widget-content ui-autocomplete ui-front" unselectable="on" style="display: none;"></ul>
      </div>
    </div>
  </div>
  <div class="header-menu">…</div>
  <div class="master-wrapper-content">…</div>
  <div class="footer">…</div>
</div>
<script src="/js/ugb2rim6qtxni-qfi-iama.scripts.js?v=IhJEl_-QC2MC4IKAv2wqLUVRXT0"></script>
<script src="/Plugins/Widgets.NivoSlider/Scripts/jquery.nivo.slider.js?v=nkJnAFgIzTVz5JJR5Ig5gsy7Gs0"></script>
```

rapper-page div.header div.header-lower div.search-box.store-search-box form#small-search-box-form input#small-searchterms.search-box-text.ui-autocomplete-input

(//input[@placeholder='Search store'])[1]

1	SH Selector	input[placeholder='Search store']
1	Rel cssSelector	input[placeholder='Search store']
1	Rel XPath	//input[@placeholder='Search store']
1	index XPath	(//input[@placeholder='Search store'])[1]
	testRigor Path	"Search store"

<input type="text" class="search-box-text ui-autocomplete-input" id="small-searchterms" autocomplete="off" name="q" placeholder="Search store" aria-label="Search store" style...

In [22]:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
import time

# Specify the path to the ChromeDriver executable
chromedriver_path = 'C:\\Users\\user\\PycharmProjects\\selenium\\day 1 selenium\\ch

# Create a Service object with the ChromeDriver executable
service_object = Service(chromedriver_path)

# Create a new instance of the Chrome driver, using the Service object
driver = webdriver.Chrome(service=service_object)

# Open the website
driver.get('https://demo.nopcommerce.com/')

# RELATIVE XPATH
driver.find_element(By.XPATH, '//input[@id="small-searchterms"]').send_keys('T-shir

time.sleep(1)

driver.find_element(By.XPATH, '//button[@type="submit"]').click()
```

What is the difference between Absolute XPath & Relative XPath ?

- **Absolute XPath:** An absolute XPath starts with the `/` symbol and specifies the complete path to the element from the root node of the HTML document. This means that the absolute XPath will always be the same, regardless of the order of the elements in the DOM.
- **Relative XPath:** A relative XPath starts with the `//` symbol and specifies the path to the element relative to the current node. This means that the relative XPath will change if the order of the elements in the DOM changes.

Here is an example of an absolute XPath:

```
/html/body/div[1]/div/div[1]/a
```

This XPath starts at the root node (`html`) and then specifies the path to the `a` element within the `div` elements. The order of the elements in the DOM does not matter, so this XPath will always point to the same element.

Here is an example of a relative XPath:

```
//a[@href='https://www.google.com']
```

This XPath starts at the current node and then specifies the path to the `a` element with the href attribute of `https://www.google.com`. The order of the elements in the DOM does not matter, so this XPath will point to the same element regardless of where it is located in the DOM

In general, relative XPaths are preferred over absolute XPaths in Selenium Python because they are more resilient to changes in the DOM. However, there are some cases where absolute XPaths may be necessary, such as when you need to locate an element that is not visible in the DOM.

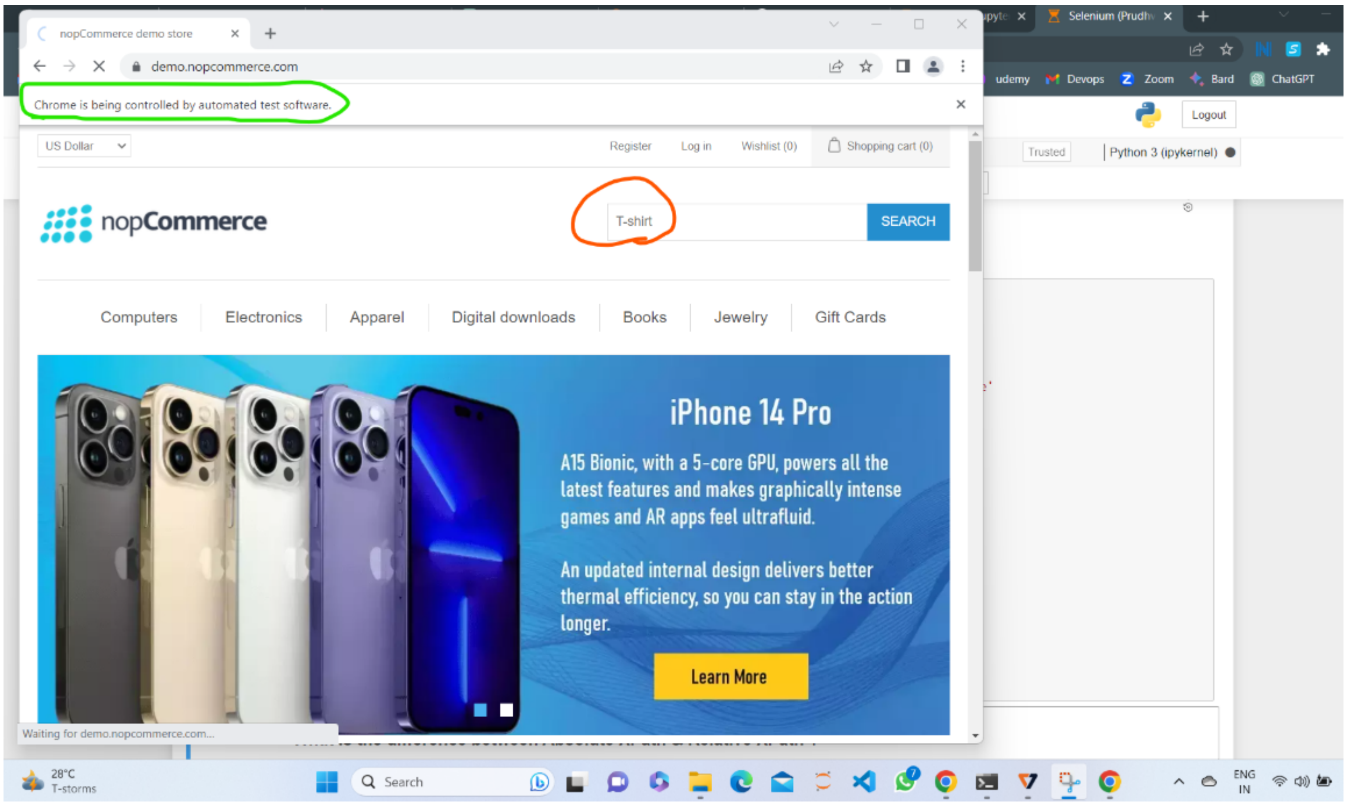
Here are some additional considerations when using absolute and relative XPaths in Selenium Python:

- **Absolute XPaths are more difficult to maintain.** If you need to change the structure of the DOM, you will also need to update the absolute XPaths that you are using.
- **Relative XPaths are more efficient.** Selenium Python can more quickly locate elements using relative XPaths than absolute XPaths.

Ultimately, the best way to choose between absolute and relative XPaths is to consider the specific needs of your automation project. If you need to locate an element that is not visible in the DOM or if you need to make sure that your XPaths are not affected by changes in the DOM, then you should use absolute XPaths. However, if you need to make your automation project more efficient, then you should use relative XPaths.

In [18]:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service

# Specify the path to the ChromeDriver executable
chromedriver_path = 'C:\\Users\\user\\PycharmProjects\\selenium\\day 1 selenium\\ch

# Create a Service object with the ChromeDriver executable
service_object = Service(chromedriver_path)

# Create a new instance of the Chrome driver, using the Service object
driver = webdriver.Chrome(service=service_object)

# Open the website
driver.get('https://demo.nopcommerce.com/')

# OR & AND
driver.find_element(By.XPATH, '//input[@id="small-searchterms" or @type="text"]').s
driver.find_element(By.XPATH, '//button[@type="submit" and @class="button-1 search-
```


nopCommerce demo store
demo.nopcommerce.com
Chrome is being controlled by automated test software.
US Dollar
Register
Log in
Wishlist (0)
Shopping cart (0)
nopCommerce
T-shirt
SEARCH
Computers
Electronics
Apparel
Digital downloads
Books
Jewelry
Gift Cards
iPhone 14 Pro
A15 Bionic, with a 5-core GPU, powers all the latest features and makes graphically intense games and AR apps feel ultrafluid.
An updated internal design delivers better thermal efficiency, so you can stay in the action longer.
Learn More
Waiting for demo.nopcommerce.com...

Here's that demonstrates the usage of OR (|) and AND (`and`) operators in XPath expressions:

In this code, after opening the website, we demonstrate the usage of the OR (|) and AND (`and`) operators in XPath expressions:

- **OR Operator**: The XPath expression `'//input[@id="username" or @name="email"]'` uses the OR operator (|) to locate an input element that has either the `id` attribute value of "username" or the `name` attribute value of "email".
- **AND Operator**: The XPath expression `'//input[@type="text" and @name="username"]'` uses the AND operator (`and`) to locate an input element that has the `type` attribute value of "text" and the `name` attribute value of "username".

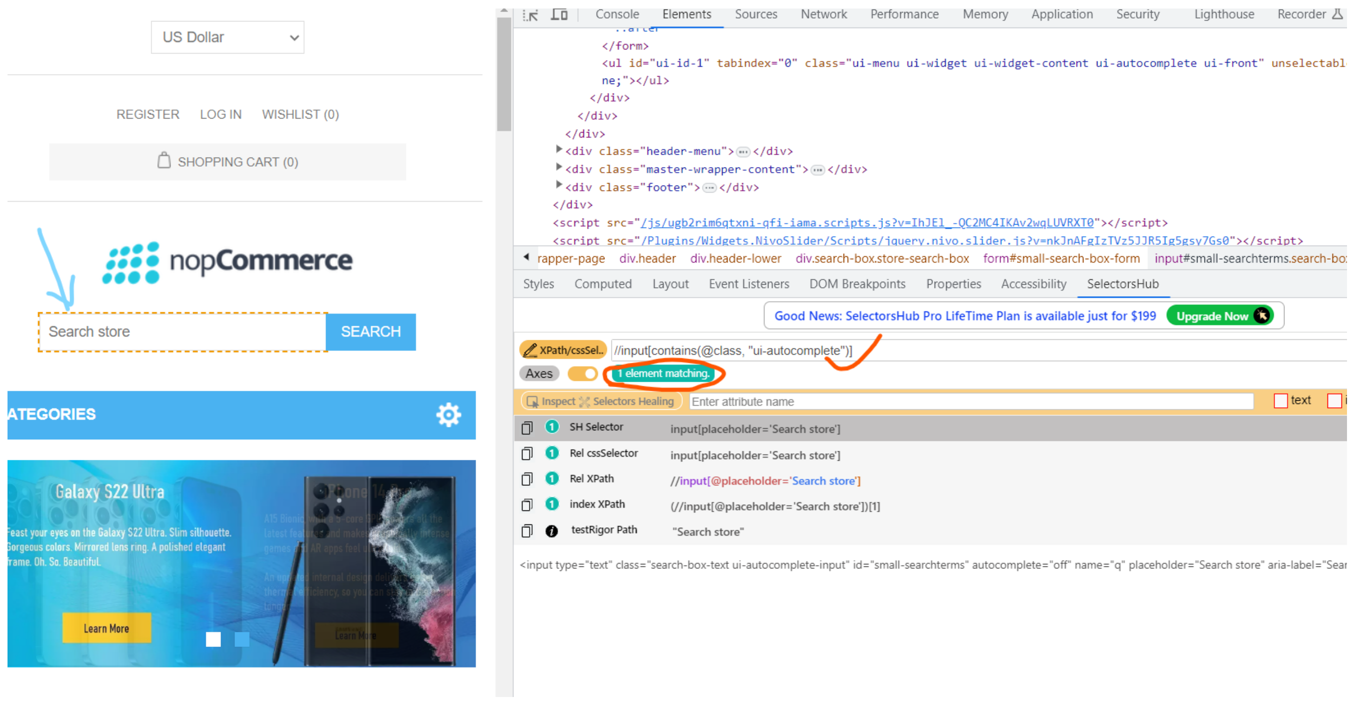
We use the `find_element()` method with the respective XPath expressions and the `By.XPATH` constant to locate the elements. Then, we print the outer HTML of each element using the `get_attribute()` method.

Finally, we close the browser using `driver.quit()` .

These operators provide flexibility in constructing XPath expressions to locate elements based on multiple conditions.

In [26]:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service

# Specify the path to the ChromeDriver executable
chromedriver_path = 'C:\\Users\\user\\PycharmProjects\\selenium\\day 1 selenium\\ch

# Create a Service object with the ChromeDriver executable
service_object = Service(chromedriver_path)

# Create a new instance of the Chrome driver, using the Service object
driver = webdriver.Chrome(service=service_object)

# Open the website
driver.get('https://demo.nopcommerce.com/')

# CONTAINS & STARTS-WITH
driver.find_element(By.XPATH, '//input[contains(@class, "ui-autocomplete")]').send_

time.sleep(5)

driver.find_element(By.XPATH, '//button[starts-with(@type, "sub")]').click()
```



```
        </form>
        <ul id="ui-id-1" tabindex="0" class="ui-menu ui-widget ui-widget-content ui-autocomplete ui-front" unselectabl
        ne;"></ul>
      </div>
    </div>
  </div>
  <div class="header-menu">…</div>
  <div class="master-wrapper-content">…</div>
  <div class="footer">…</div>
</div>
<script src="/js/ugb2rim6qtxni-qfi-iama.scripts.js?v=IhJEl_-QC2MC4IKAv2wqLUVRXT0"></script>
<script src="/Plugins/Widgets.NivoSlider/Scripts/jquery.nivo.slider.js?v=nkJnAFgIzTVz5JJR5Ig5gsy7Gs0"></script>
```

rapper-page div.header div.header-lower div.search-box.store-search-box form#small-search-box-form input#small-searchterms.search-bo

//input[contains(@class, "ui-autocomplete")]

1 element matching.

1	SH Selector	input[placeholder='Search store']
1	Rel cssSelector	input[placeholder='Search store']
1	Rel XPath	//input[@placeholder='Search store']
1	index XPath	(//input[@placeholder='Search store'])[1]
	testRigor Path	"Search store"

<input type="text" class="search-box-text ui-autocomplete-input" id="small-searchterms" autocomplete="off" name="q" placeholder="Search store" aria-label="Sear

Here's a brief explanation of the `contains()` and `starts-with()` functions commonly used in XPath expressions:

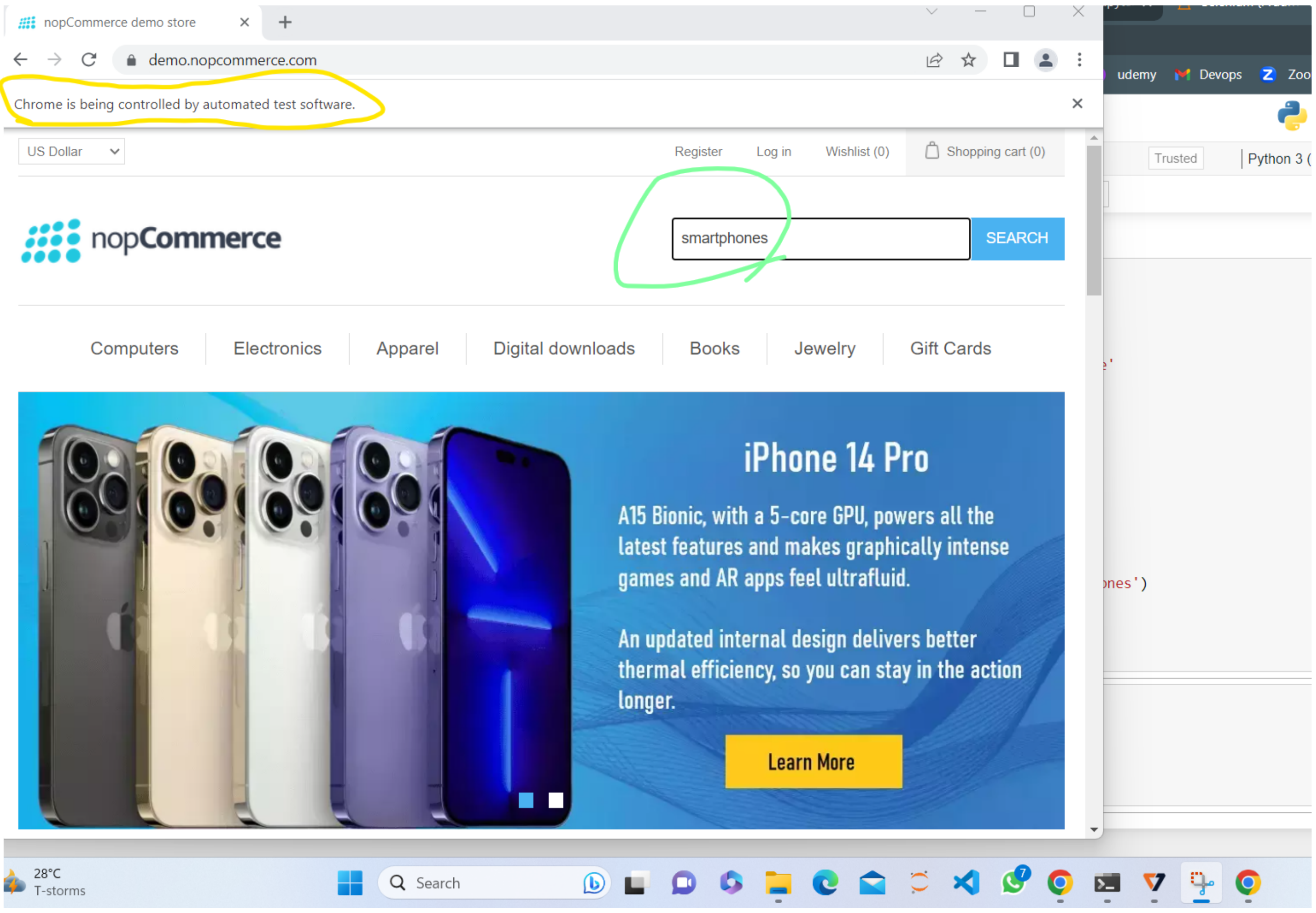
- **Contains**: The `contains()` function in XPath allows you to search for elements that contain a specified value within a particular attribute. It is useful when you want to locate elements based on partial attribute values.

For example, the XPath expression `//input[contains(@class, "ui-autocomplete")]` will locate an `<input>` element that has a `class` attribute containing the substring "ui-autocomplete". It matches elements with classes like "ui-autocomplete-textbox" or "ui-autocomplete-input".

- **Starts-with**: The `starts-with()` function in XPath allows you to search for elements that have attribute values starting with a specified substring. It is useful when you want to locate elements based on the initial part of their attribute values.

For example, the XPath expression `//button[starts-with(@type, "sub")]` will locate a `<button>` element that has a `type` attribute starting with the substring "sub". It matches elements with types like "submit" or "subscribe".

[illegible]


Chrome is being controlled by automated test software.

iPhone 14 Pro

A15 Bionic, with a 5-core GPU, powers all the latest features and makes graphically intense games and AR apps feel ultrafluid.

An updated internal design delivers better thermal efficiency, so you can stay in the action longer.

In [ ]: