**Selenium WebDriver:** refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.

Selenium WebDriver is a [W3C Recommendation](https://www.w3.org/TR/webdriver1/)

* WebDriver is designed as a simple and more concise programming interface.
* WebDriver is a compact object-oriented API.
* It drives the browser effectively.

Webdriver is API(Application Programming Interface):

**Architecture of webdriver:**

**Selenium 3.x**: w3c(world consercative) rules



This image tells us about the core selenium webdriver architecture and the major selenium components which comprise WebDriver.

* ***Selenium WebDriver Client Libraries / Language Bindings****–* Software Testers want to select languages that they are comfortable with. Since WebDriver Architecture supports different languages, so there are bindings available for a range of languages like [Java**,**](https://www.toolsqa.com/java/java-tutorial/) C#, [Python**,**](https://www.toolsqa.com/python-tutorial/) Ruby, PHP, etc. Anyone who has a basic knowledge of working with any programming language can get specific language bindings and can start off. This is how Selenium Achitecture provides flexibility to testers to do automation in their comfort zone.
* ***JSON WIRE PROTOCOL****-* As per the Selenium Architecture above, the JSON Wire Protocol facilitates all the communication that is happening in Selenium between the browser and the code. This is the heart of Selenium. JSON Wire Protocol provides a medium for data transfer using a RESTful (Representational State Transfer)  API which provides a transport mechanism and defines a RESTful web service using  JSON over HTTP.
* ***Browser Drivers****–* Since there are various browsers that are supported by Selenium, each browser has its own implementation of the W3C standard that Selenium provides. As such browser-specific binaries are available that are specific to the browser and hides the implementation logic from the end-user. JSONWire protocol establishes a connection between the browser binaries and the client libraries.
* ***Browsers****–* Selenium will be only able to run tests on the browsers if they are locally installed, either on the local machine or on the server machines. So browser installation is necessary.

**Selenium 4.x:**

**Jason replaced with w3c protocol**

**3.x**: selenium language bindings---**json wire protocol**—browser drivers—w3c—browsers

**4.x**: selenium language bindings---**w3c**—browser drivers—w3c—browsers

**Task1:**

**Open a browser and run any url with automation:**

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

service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')

driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://www.facebook.com/')………..**URL PATH TO OPEN**

driver.maximize\_window() **………..MAXIMISE THE BROWSER WINDOW**

time=time.sleep(2)….**OPEN BROWSER TIME GIVEN TO SEE WHAT HAPPEN**

print(driver.title) ….**GET THE TITLE OF URL PAGE**  
driver.close() **….CLOSE THE BROWSER**

**LOCATORS:**

Purpose of locators is **Identify the elements.**

Locator is a command that tells Selenium IDE which GUI elements ( say Text Box, Buttons, Check Boxes etc) its needs to operate on.

Locating elements in Selenium WebDriver is performed with the help of findElement() and findElements() methods provided by WebDriver and WebElement class.

* **findElement()** returns a WebElement object based on a specified search criteria or ends up throwing an exception if it does not find any element matching the search criteria.
* **findElements()** returns a list of WebElements matching the search criteria. If no elements are found, it returns an empty list.

| **sl.no.** | **findElement()** | **findElements()** |
| --- | --- | --- |
| 1 | It returns the first web element which matches with the locator. | It returns all the web elements which match with the locator. |
| 2 | Syntax − WebElement button = webdriver.findElement(By.name("<<Name value>>")); | Syntax − List<WebElement> buttons = webdriver.findElements(By.name("<<Name value>>")); |
| 3 | NoSuchElementException is thrown if there are no matching web elements | Empty list is returned if there are no matching elements. |
| 4 | Send keys apperered | Send key not apperered |
| 5 | Element is single web element object | Element is list collection object |

**Case1: Having single element**:

name=driver.**find\_element**(By.XPATH,'//input[@id="email"]')  
name.send\_keys('vijay')  
  
name=driver.**find\_elements**(By.XPATH,'//input[@id="email"]')  
print(len(name))------1  
name[0].send\_keys('vijay')-----using indexing fetch the element

**case 2: having multiple elements**

footer=driver.**find\_element**(By.XPATH,'//div[@id="pageFooter"]//a')----42 elements having actually but text only first one out of it  
print(footer.text)------- मराठी

footer=driver.**find\_elements**(By.XPATH,'//div[@id="pageFooter"]//a')  
print(len(footer))-----42  
print(footer[15].text)------fetch 15 no element from footerpage

for ele in footer:  
 print(ele.text)----------fetch all elements from footerpage

**QUE 1:** **Types of locators used in Selenium** Python are as follows:

1. ID
2. Name
3. Class Name
4. Tag Name
5. Link Text
6. Partial Link Text
7. CSS Selector
8. XPath

**By.ID**

With this strategy, the first element with the id attribute value matching the location will be returned. If no element has a matching id attribute, a NoSuchElementException will be raised.

Syntax:

**driver.find\_element(By.ID, "value of id")**

driver.find\_element(By.ID,'small-searchterms').send\_keys('Vintage Style Engagement Ring')

**QUE: single web elements locators?**

**ID, NAME, LINKTEXT,PARTIALLINK TEXT**

### By.NAME

With this strategy, the first element with the name attribute value matching the location will be returned. If no element has a matching name attribute, a NoSuchElementException will be raised.

**Syntax:**

**driver.find\_element(By.NAME, "value of name")**

driver.find\_element(By.NAME,'q').send\_keys('Vintage Style Engagement Ring')

### By.CLASS\_NAME

With this strategy, the first element with the matching class attribute name will be returned. If no element has a matching class attribute name, a NoSuchElementException will be raised.

**Syntax:**

**driver.find\_element(By.CLASS\_NAME, "value of class")**

sliders=driver.find\_elements(By.CLASS\_NAME,'a-carousel-col a-carousel-right celwidget')

print(len(sliders))

### By.LINK\_TEXT

With this strategy, the first element with the link text value matching the location will be returned. If no element has a matching link text attribute, a NoSuchElementException will be raised.

**Syntax:**

**driver.find\_element(By.LINK\_TEXT, "Text of Link")**

driver.find\_element(By.LINK\_TEXT,'Register').click()

### By.PARTIAL\_LINK\_TEXT

With this strategy, the first element with the partial link text value matching the location will be returned. If no element has a matching partial link text attribute, a NoSuchElementException will be raised.

Syntax:

**driver.find\_element(By.PARTIAL\_LINK\_TEXT, "partial Text of Link")**

driver.find\_element(By.PARTIAL\_LINK\_TEXT,'Reg').click()

**QUE 2:** difference between link text and partial link text

### By.TAG\_NAME

With this strategy, the first element with the given tag name will be returned. If no element has a matching tag name, a NoSuchElementException will be raised.

**Syntax:**

**driver.find\_element(By.TAG\_NAME, "Tag name")**

links=driver.find\_elements(By.TAG\_NAME,'a')

print(len(links))

**QUE: How to find multiple web elements?**

**- Using class name and tag name will find elements**

### By.CSS\_SELECTOR

With this strategy, the first element with the matching CSS selector will be returned. If no element has a matching CSS selector, a NoSuchElementException will be raised.

**Syntax:**

**driver.find\_element(By.CSS\_SELECTOR, "CSS Selectors")**

**COMBINATION OF CSS SELECTOR:**

**1.TAG AND ID**:

Syntax: **tagname#value of id OR #VALUE OF ID**

driver.find\_element(By.CSS\_SELECTOR,'input#email').send\_keys('vijay')

**OR**  
driver.find\_element(By.CSS\_SELECTOR,'#email').send\_keys('vijay')

**2.TAG AND CLASS:**

Syntax: **TAGNAME.VALUE OF CLASS….OR .VALUE OF CLASS**

driver.find\_element(By.CSS\_SELECTOR,'input.inputtext').send\_keys('vijay')

**OR**  
driver.find\_element(By.CSS\_SELECTOR,'.inputtext').send\_keys('vijay')

**NOTE:** Sometime class value not taken due to space is available in class value so that time remove all next to space from class value and used then its work fine

**3. TAG AND ATTRIBUTES:**

Syntax: **Tagname[attributes=value] OR [attributes=value]**

driver.find\_element(By.CSS\_SELECTOR,'input[autocomplete=username]').send\_keys('vijay')

**OR**  
driver.find\_element(By.CSS\_SELECTOR,'[autocomplete=username]').send\_keys('vijay')

**4. TAG, CLASS AND ATTRIBUTES:**

Syntax: **TAGNAME.VALUE OF CLASS[attributes=value] OR**

**.VALUE OF CLASS[attributes=value]**

driver.find\_element(By.CSS\_SELECTOR,'input.inputtext[autocomplete=username]').send\_keys('vijay')

**or**  
driver.find\_element(By.CSS\_SELECTOR,'.inputtext[autocomplete=username]').send\_keys('vijay')

#### **By.XPATH**

With this strategy, the first element with pattern of xpath matching the location will be returned. If no element has a matching element attribute, a NoSuchElementException will be raised.

**Syntax:**

**driver.find\_element(By.XPATH, "xpath")**

-xpath is defined as XML path

-it is a syntax or language for finding any element on the web page using XML path expression

-Xpath is used to find the location of any elements on a webpage using HTML DOM structure

-Xpath can be used to navigate through elements and attributes in DOM(is an API interface provided by browser, when web page is loaded, the browser creates a document object model of the page)

-Xpath is an address of the element.

**Types of X-path**

There are two types of XPath:

**1) Absolute XPath**

**2) Relative XPath**

#### **Absolute XPath**:

It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed.

The key characteristic of XPath is that it begins with the single forward slash(/) ,which means you can select the element from the root node.

/html/body/div[2]/div[1]/div/h4[1]/b/html[1]/body[1]/div[2]/div[1]/div[1]/h4[1]/b[1]

**Relative Xpath**:

starts from the middle of HTML DOM structure. It starts with double forward slash (//). It can search elements anywhere on the webpage, means no need to write a long xpath and you can start from the middle of HTML DOM structure. Relative Xpath is always preferred as it is not a complete path from the root element.

**Syntax: //tagname[@attribute=’value’] or**

**//\*[@attribute=’value’ …if tagname N/A**

**QUE: Difference between xpths types**

|  |  |  |
| --- | --- | --- |
| **Point of Difference** | **Absolute Path** | **Relative Path** |
| **Starts with** | **Single Forward Slash**. Select the element from the root <html> and cover the whole path to the element. It is also known as **complete or Full Xpath**. | **Double Forward Slash**. Expression can starts in the middle of the HTML DOM structure. |
| **Speed** | **Faster.** It identify the element very fast. | **Slower** compare to absolute. it will take more time in identifying the element as we specify the partial path not (exact path). |
| **Failure Chances** | **More**. It Changes Frequently, if there are any changes made in the path of the element then XPath gets failed. | Failure chance of well written relative path is **very less** |
| **Example** | **/html/head/body/form/table**  **/tbody/tr/th** if any tag will be added before table the path will fail. | **//table/tbody/tr/th**  Doesn't matter if anything added before table. |
| **Used only** | **Tags/nodes** | **Use attributes also** |

**QUE: How to write Xpath manually:**

**Absolute**: html/body/div[6]/div[1]/div[1]/div[2]/div[1]/ul/li[1]/a

**Relative Xpath**:

**Syntax://tagname[@attribute=’value’]**

//input[@id='small-searchterms']

**QUE: How to write Xpath automatically:**

* 1. Right click on html path---copy---copy xpath/copy full xpath (browser inbuilt path)
  2. Selectorshub

**QUE: which xpath mostly used/preferred:**

Relative xpath preferd

**Why:**

1.More stable than absolute xpath---if developer done some changes or introduce new element then absolute xpath broken

2.if developer change the location then absolute xpath broken

**Xpath options:**

**1.Xpath with OR:**

name=driver.find\_element(By.XPATH,'//input[@name="name" or @type="text"]').send\_keys('vijay')

**2.Xpath with AND:**

name=driver.find\_element(By.XPATH,'//input[@name="name" and @type="text"]').send\_keys('vijay')

**FINDING DYNAMIC WEB ELEMENTS THEN USED:**

<input class="form-control" id="exampleInputPassword1" placeholder="Password" type="password">

**3.contains: :**

**//tagname[contains(@attributes,’value of attribute at any position’)]**

Any word matched from any place then its show that elelments.

name=driver.find\_element(By.XPATH,"//input[contains(@type,'word')]").send\_keys('vijay')

**4.starts with:**

**//tagname[starts-with(@attributes,’value of attribute at start position’)]**

Match word must be at the start position then only find elements otherwise show exception

name=driver.find\_element(By.XPATH,"//input[starts-with(@type,'pass')]").send\_keys('vijay')

**5.text:**

Syntax**: //tagname[text()=’text value’]**

if we want to text web element then text function used

text=driver.find\_element(By.XPATH,"//a[text()='Sell']").click()

**XPATH AXES:**

To navigate the hierarchical tree of nodes in an XML document, XPath uses the concept of axes. The XPath specification defines a total of 13 different axes that we will learn in this section.

A list of 13 XPath axes methods in Selenium WebDriver is as follows:

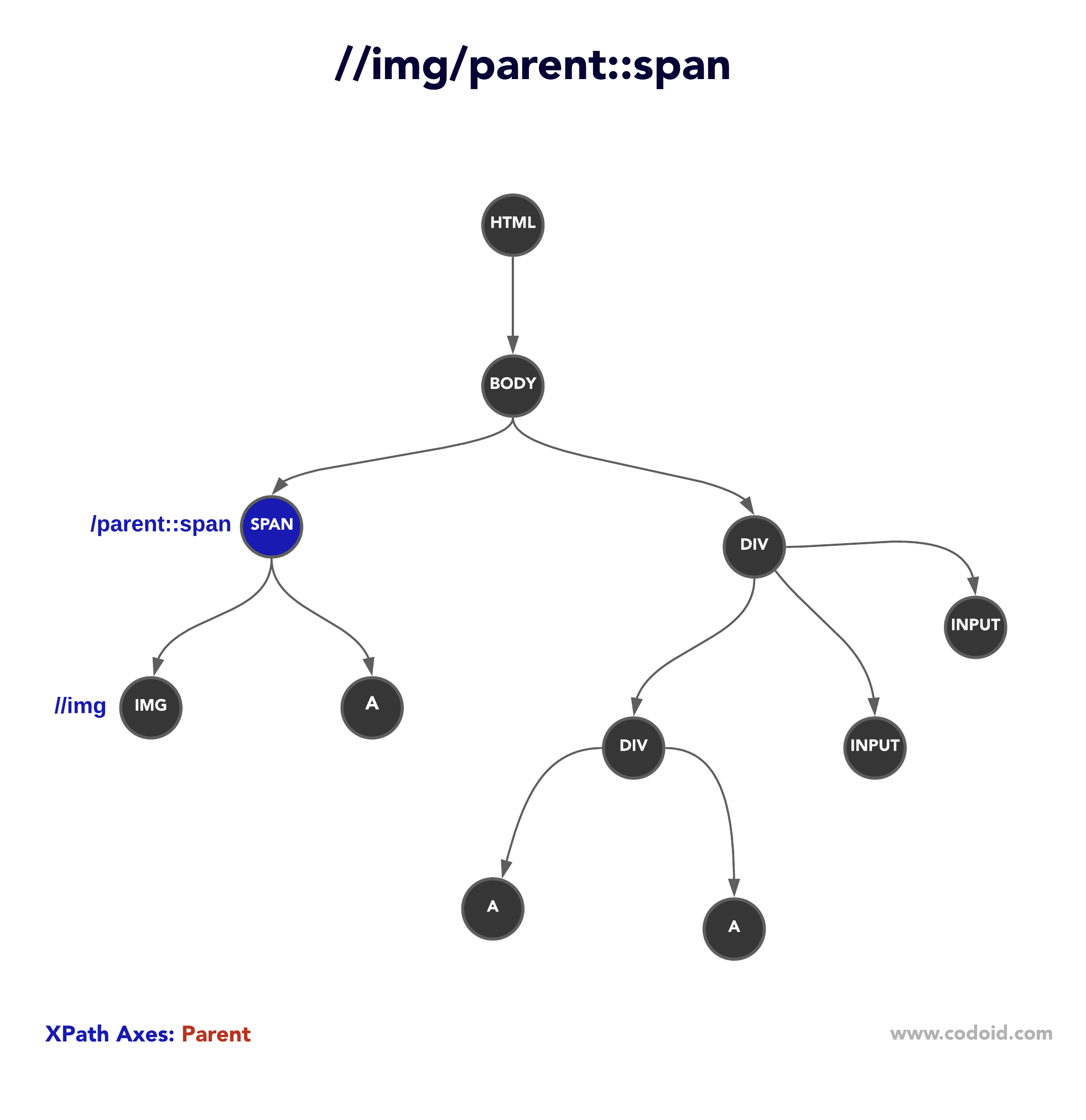
1. Child Axis
2. Parent Axis
3. Self Axis
4. Ancestor Axis
5. Descendant Axis
6. Following Axis
7. Following-sibling Axis
8. Preceding Axis
9. Preceding-sibling Axis

Each axis contains various nodes that depend on the current node. An XPath axis is a collection of nodes that satisfy the current navigation criteria.

**Parent:**

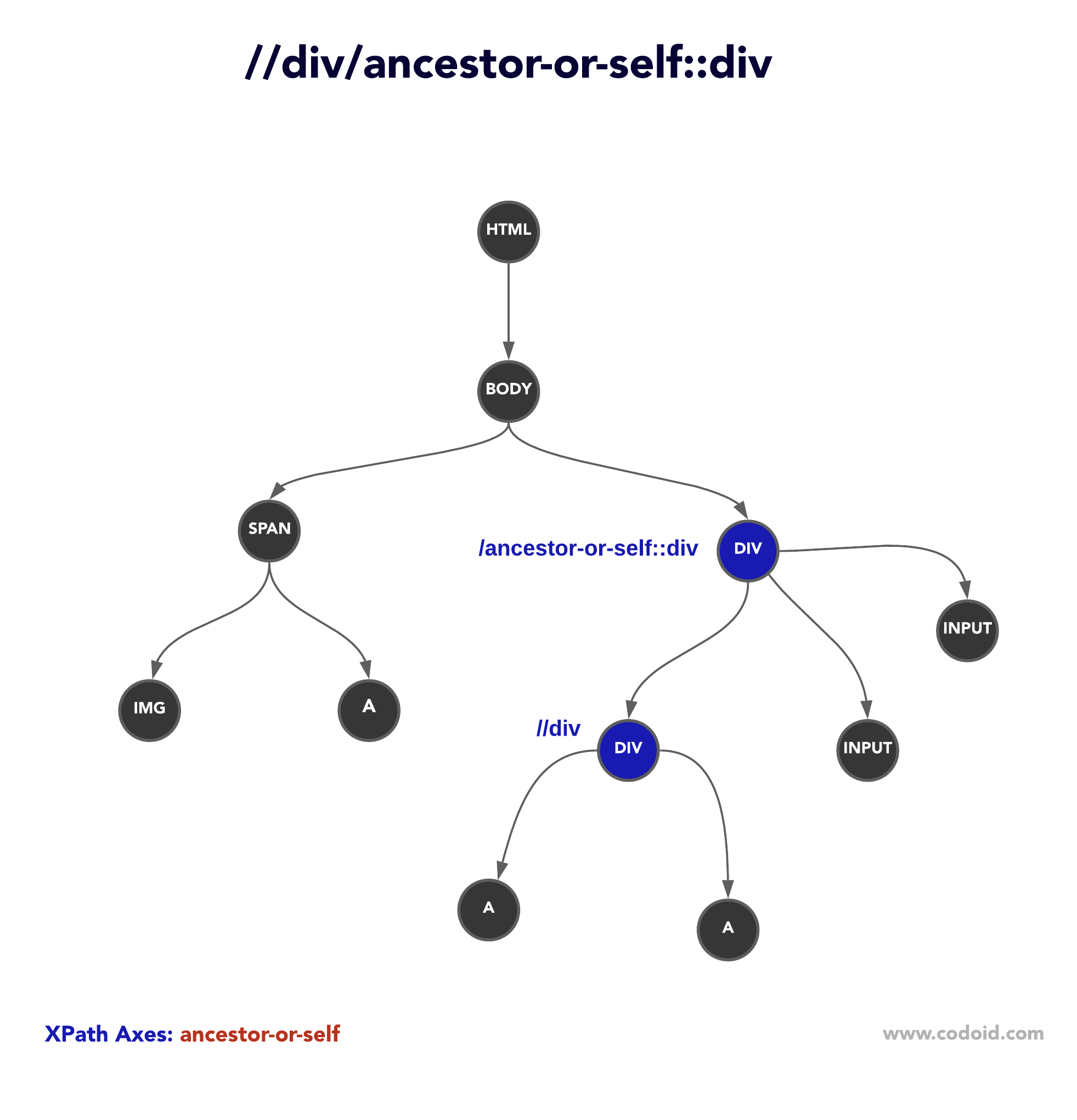
If you want to select the parent node from the child, use the parent axis.

Syntax: **//\*[attributes=’value’]/parent::tagname**

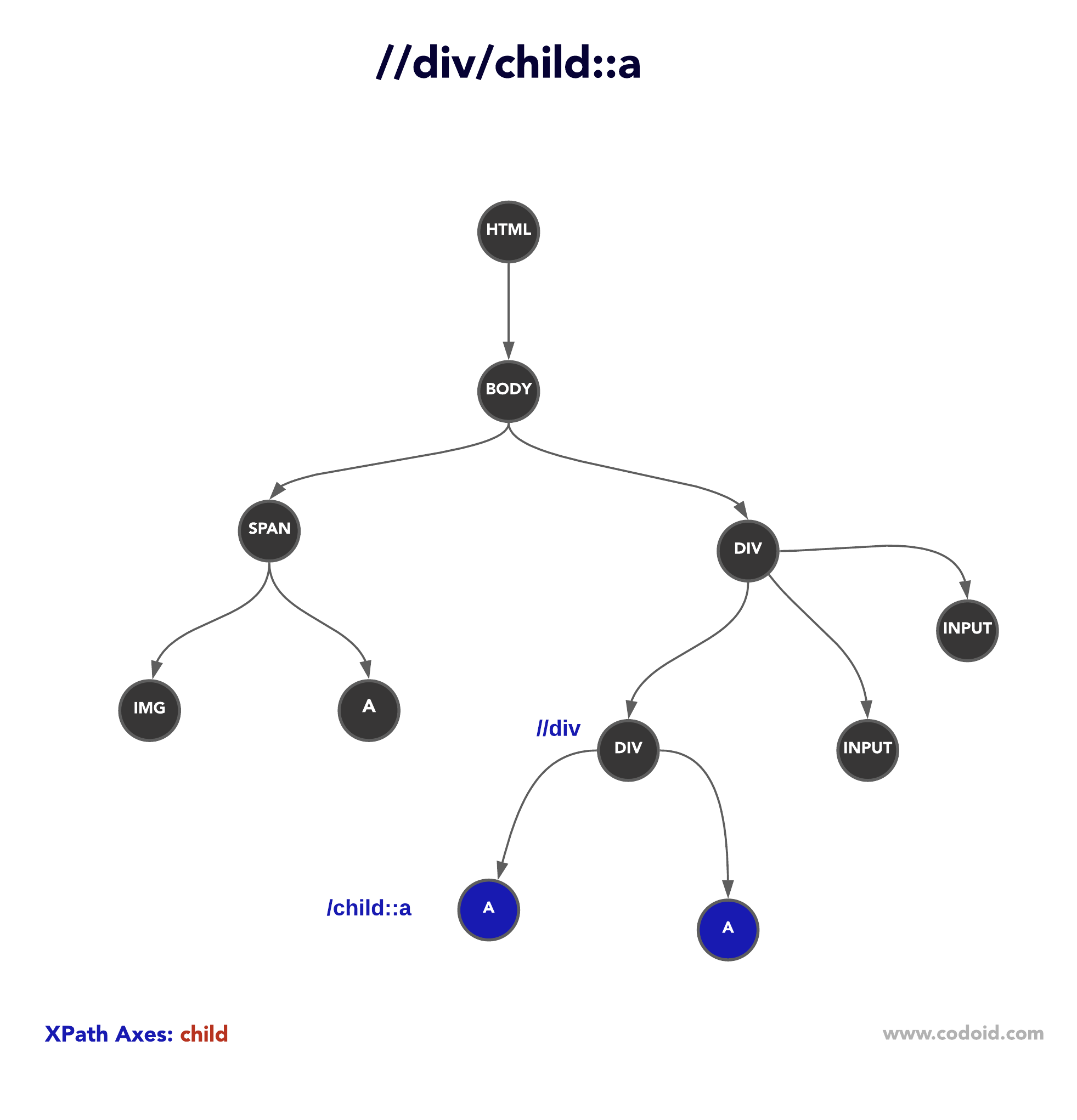


**ancestor-or-self:**

ancestor-or-self axis selects the parent, ancestors, and the context node as well. Syntax: **//\*[attributes=’value’]/ancestor::tagname**



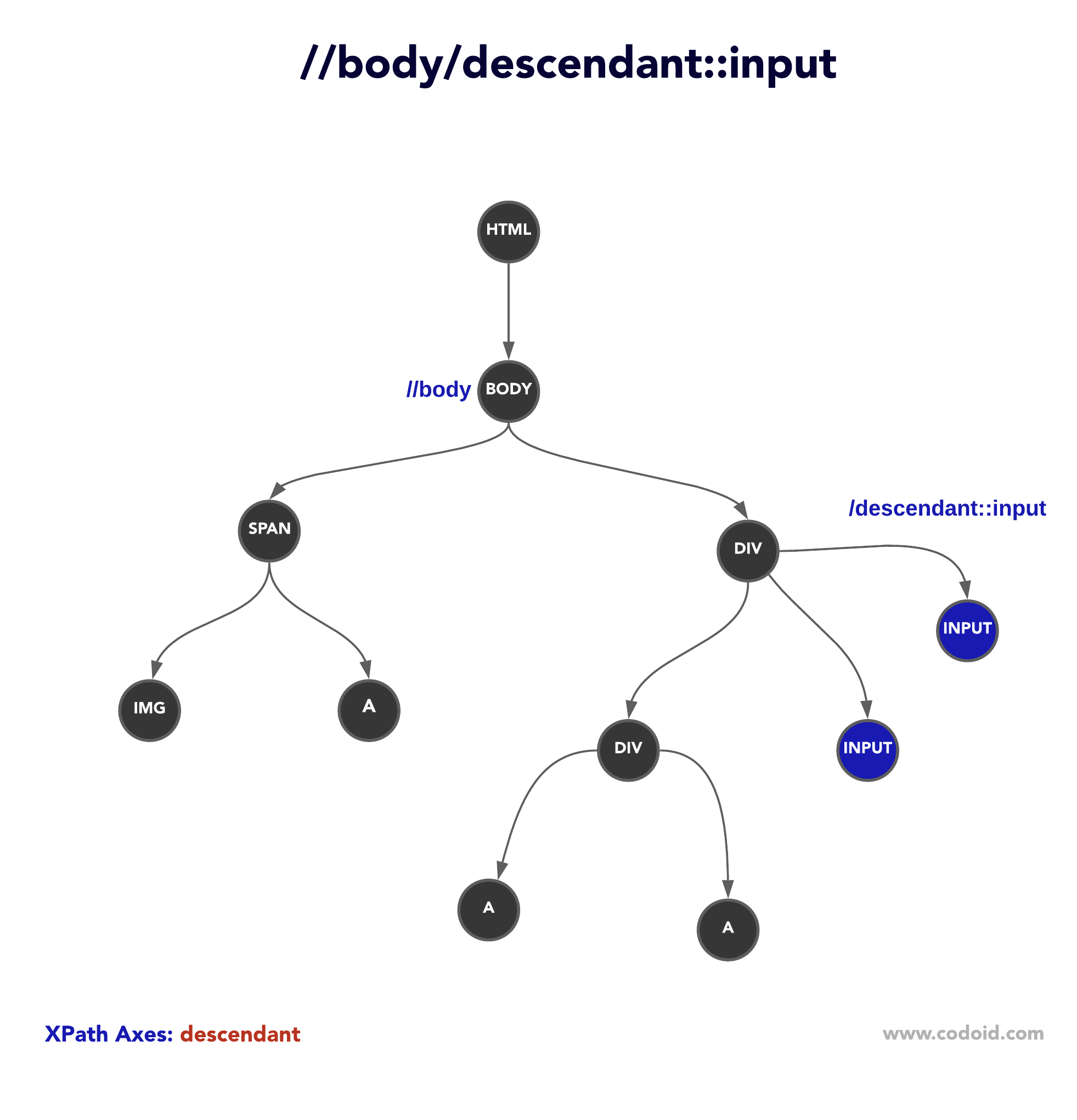
**Child:**To select all the children nodes of the context node, you can use the child axis. Syntax: **//\*[attributes=’value’]/child::tagname**



**Descendant:**

To select children and the descents nodes, use the descendant axis. The descendant axis is a widely used axis by automation testers.

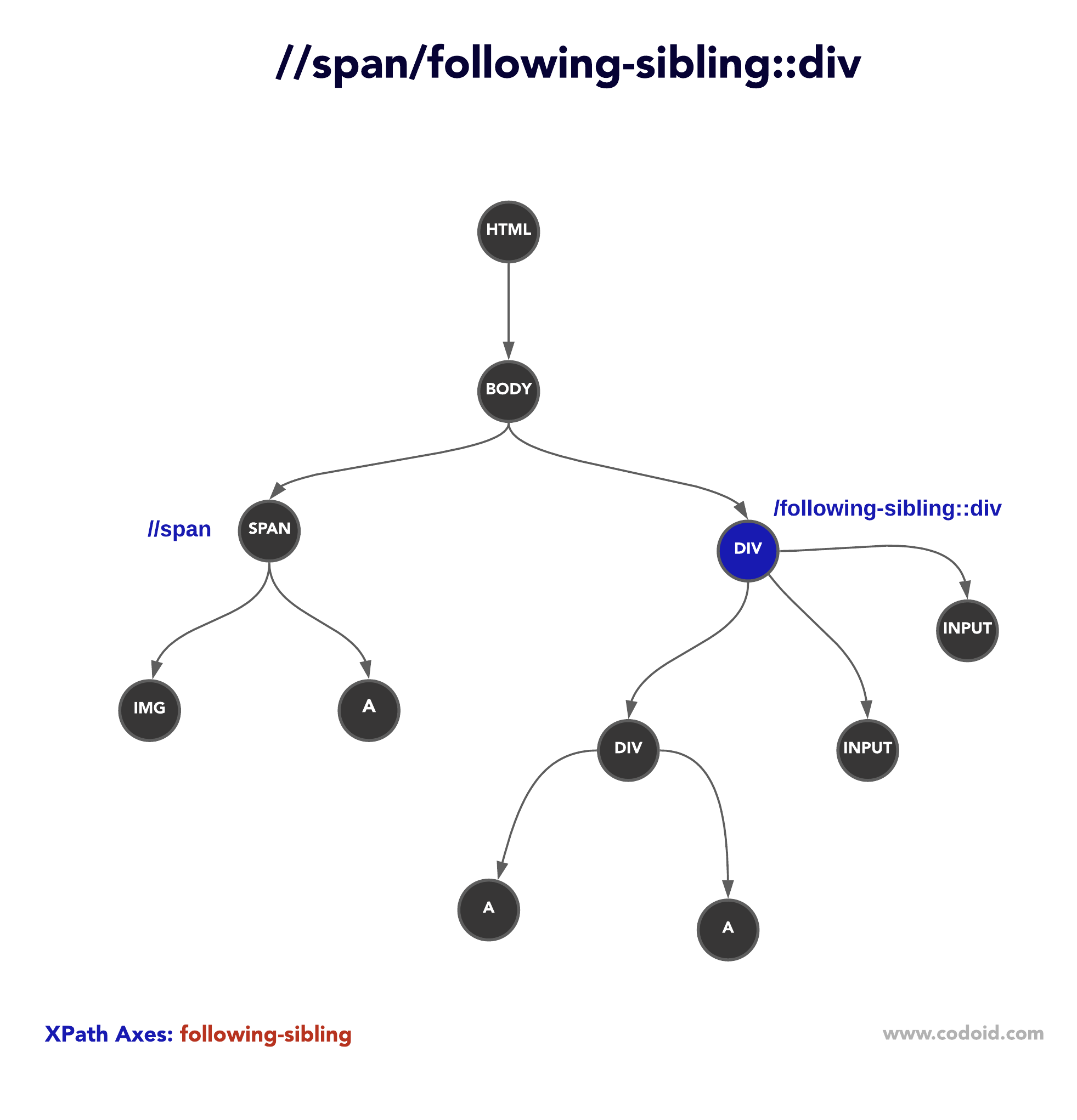
Syntax: **//\*[attributes=’value’]/descendant::tagname**



**following-sibling:**

If you would like to select siblings that are after the current node, try the following-sibling axis.

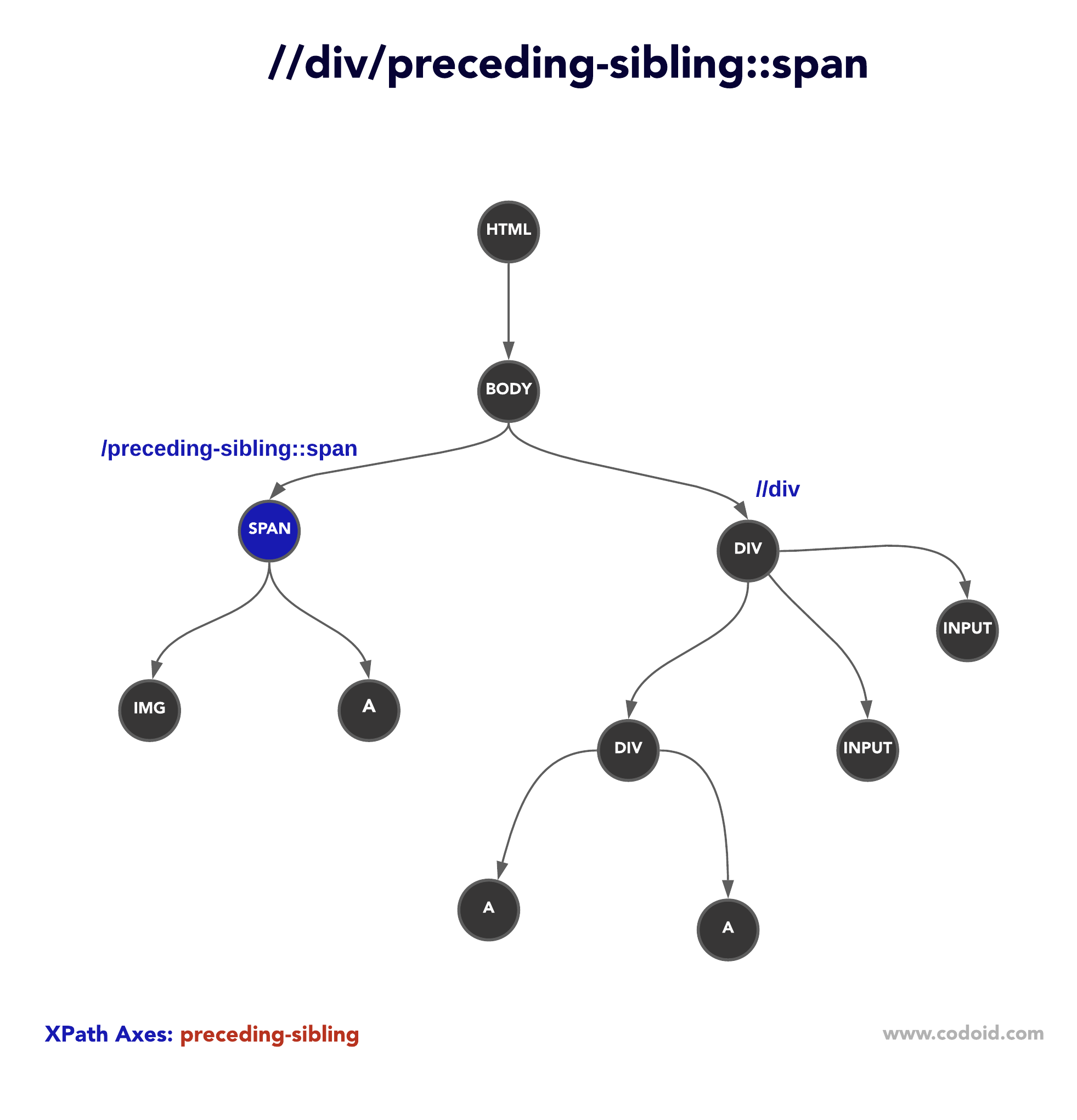
Syntax**: //current html tag[@attributes=’value’]/following-sibling::sibling tag [@attribute=’value’]**



**preceding-sibling:**

If you would like to select siblings that are before the current node, try the preceding-sibling axis.

Syntax: **//current html tag[@attributes=’value’]/preceding-sibling::previous tag [@attribute=’value’]**



**Following:**

Syntax: **//\*[attributes=’value’]/following::tagname**

**Preceding:**

Syntax: **//\*[attributes=’value’]/preceding::tagname**

**Webdriver commands:**

1. **Browser Commands**

* **Get** command  -----opening the application url
* **Get Title** command-------to capture the title of the current webpage
* **Get Current URL** command-----to capture the current url of the web page
* **Get Page Source** Command-------to capture source code of thew web page
* **Close** command-------------close the application url
* **Quit** command-------------close the all the application

**QUE**: what is the difference between close and quit commands:

* The close() method can close the browser in focus.
* The close() method closes the present window on which we are working like parent window close(where driver focused)
* While quit() method works with the driver.dispose() method that closes every successive window.
* While quit() method suspends all the driver sessions and instances, thereby closing each opened window(multiple window closed)

2. **Navigation Commands**

Navigate to command from one browser to another browser

* Back command---------- driver.back()
* Forward command------- driver.forward()
* Refresh commands----- driver.refresh()

3. **WebElement Commands**

* Clear command
* Click command: used to operate the button or check box

text=driver.find\_element(By.XPATH,"//a[text()='Sell']").click()

**QUE**: Difference between text and get-attribute:

* **GetText command:**
* The getText() method **returns the innerText** of an element, that is the text which is visible on the page along with its sub-elements.
* Inner text is the text that is between the opening and closing tags. getText() ignores all the leading and trailing spaces.
* footer=driver.find\_element(By.XPATH,'//div[@id="pageFooter"]//a')  
  print(footer.text)---------- मराठी
* **getAttribute command:**
* The getAttribute() method **fetches the text contained by an attribute in an HTML** document. It returns the value of the HTML element's attribute as a string.
* If a value is not set for an attribute, it will return a NULL value. For attributes with Boolean values, getAttribute() will return either "True" or NULL. The attribute is passed as a parameter to the method.
* footer=driver.find\_element(By.XPATH,'//div[@id="pageFooter"]//a')  
  print(footer.get\_attribute('values))------none at value we must provide any attributes which value we want to fetch from web page from html
* gefTagName command
* **SendKeys command**: used for provide the input for the web element

driver.find\_element(By.CSS\_SELECTOR,'#email').send\_keys('vijay')

* getSize command
* GetCssValue command
* getLocation command
* **Submit command:**
* its works like enter key on keyboard, we get the web element then bowser have function to go direct with enter key that time we used submit command. Google search button considered for example
* driver.find\_element(By.XPATH,'//textarea[@id="APjFqb"]').send\_keys('seleniu')  
  driver.find\_element(By.XPATH,'//textarea[@id="APjFqb"]').submit()
* open the information regarding the selenium on google

**Conditional command:**

* **IsSelected** command: for radio buttons and check box used

rad\_male=driver.find\_element(By.XPATH,'//input[@id="gender-male"]')  
radfemale=driver.find\_element(By.XPATH,'//input[@id="gender-female"]')

print(rad\_male.is\_selected()) false  
print(radfemale.is\_selected()) false  
  
rad\_male.click()  
print('after selection:',rad\_male.is\_selected()) true

* **IsEnabled command:**
* **IsDisplayed command:**

name=driver.find\_element(By.XPATH,'//input[@id="email"]')

print('display status:', name.is\_displayed())  
print('enable status:', name.is\_enabled())

output: display status: True

enable status: True

**Commands practical:**

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
*# from selenium.webdriver.common.by import By*import time  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
**driver.get**('https://www.facebook.com/login/') **opening application URL**

print(**driver.title**) **Log in to Facebook**

print(**driver.current\_url**) **https://www.facebook.com/login/**

print(**driver.page\_source**) **display the html page source all**time.sleep(2)

**driver.close() close the current working application URL**

**driver.quit() close all application on web page**

**WAIT COMMANDS:**

at the time of synchronisation problem that time used

**1.implicit wait**

**2.explicit wait**

**QUE: Difference betweem implicit and explicit wait:**

* **Timeouts**: Implicit wait sets a default timeout for the entire test runtime, while explicit wait sets timeouts for specific conditions.
* **Condition**: Implicit wait waits for an element to appear on the page, while explicit wait waits for a specific condition, such as the presence of an element or the element to be clickable.
* **Scope**: Implicit wait applies globally, while explicit wait applies locally to a specific element.
* **Exception**: Implicit wait throws a *NoSuchElementException* when the *WebDriver* cannot find the element within the specified timeout. In contrast, explicit wait throws a *TimeoutException* when the element doesn't meet the condition within the specified timeout.

Implicit wait is helpful when we want to wait for a certain amount of time for elements to appear on the page. However, explicit wait is the better option if we need to wait for a specific condition.

**Explicit practice:**

from selenium import webdriver  
from selenium.common.exceptions import NoSuchElementException,ElementNotVisibleException,ElementNotSelectableException……………..**exception handling purpose**  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
import time  
from selenium.webdriver.support.wait import WebDriverWait  
from selenium.webdriver.support import expected\_conditions as EC ….**condition purpose**

service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
mywait=WebDriverWait(driver,10)……..**basic format**  
mywait=WebDriverWait(driver,10,poll\_frequency=2,ignored\_exceptions=[NoSuchElementException,  
 ElementNotVisibleExceptio,ElementNotSelectableException,Exception])………………..**exception handling by itself**  
  
driver.get('https://www.google.com/')  
  
searchbox=driver.find\_element(By.NAME,'q')  
searchbox.send\_keys('selenium')  
searchbox.submit()  
  
searchlink=mywait.until(EC.presence\_of\_element\_located((By.XPATH,"//h3[text()='selenium']")))………..**syntax for explicit wait find element not used**

searchlink.click()

driver.quit()

**que: why time is required in explicit wait?**

If we not provide the time then it will wait so longer up to condition is not meet, then program is stop there. so avoid this we provide time to wait i.e cut off time if condition is not meet then its exit from there.

**CHECKBOX WEB ELEMENT**: for loop necessary here

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
import time  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://itera-qa.azurewebsites.net/home/automation')  
driver.maximize\_window()

**case1: select specific checkbox**

monday=driver.**find\_element**(By.XPATH,'//input[@id="monday"]').click()

**case 2: select multiple checkbox:**

days=driver.**find\_elements**(By.XPATH,"//input[@type='checkbox' and contains(@id,'day')]")  
print(len(days))  
  
for day in days:------------**select all checkbox at once**  
 day.click()

for i in range(len(days)):------------**select all checkbox at once**  
 days[i].click()  
  
for day in days:---------**select specific checkbox from list of checkbox**  
 name=day.get\_attribute('id')  
 if name=='monday' or name=='friday':  
 day.click()  
  
for i in range(len(days)-2,len(days)):----**select only last 2 checkbox**  
 days[i].click()  
  
for i in range(len(days)):----------**select onlt top 2 checkbox**  
 if i<2:  
 days[i].click()

time.sleep(2)  
driver.close()

**LINKS:** tagname: a property:href

**Types of links**:

1. internal link: navigate on same web page

2. external link: go to other web page

3. broken link: for developer used---dosen’t have direct page

Operation performed on links is:

1.**clicks on link using**:

1. linktext-------mostly preferered

2. partial linktext

2. **find no link on page and print**:----using tagname & for loop

links=driver.find\_elements(By.TAG\_NAME,'a')  
 print(len(links))

for link in links:

print(link.text)

**QUE**: tell the tagnames for different web element like image,dropdown,checkbox etc

**DROPDOWN ELEMENT: TAGNAME:SELECT**

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
**from selenium.webdriver.support.select import Select**  
import time  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://www.globalsqa.com/demo-site/select-dropdown-menu/')  
driver.maximize\_window()  
  
dropdown=**Select**(driver.find\_element(By.XPATH,'(//select)[1]'))---module

**case 1: capture single option in dropdown:**

dropdown.select\_by\_visible\_text('India')---**give any parameter from dropdown**  
dropdown.select\_by\_value('BLZ')------**value of value given from html**  
dropdown.select\_by\_index(5)----------**index value calculated and given**  
  
time.sleep(2)  
driver.close()

**case 2: capture all option in dropdown**

alloption=dropdown.**options**

print('total no of options:',len(alloption))-----249  
for country in alloption:---------list display all options available  
 print(country.text)

**QUE:** How to select dropdown option without using built function like value,index..

for country in alloption:------------**using for loop anf if logic** if country.text=='India':  
 country.click()  
 break

**Frames/Iframes:**

-**switch\_to.frame (name of the frame)**

**-switch\_to.frame (id of the frame)**

**- switch\_to.frame (webelement)**

**- switch\_to.frame (0) indexing**

**- switch\_to.default\_content()**

**iFrame in Selenium Webdriver:** is a web page or an inline frame which is embedded in another web page or an HTML document embedded inside another HTML document.

The iframe is often used to add content from other sources like an advertisement into a web page. The iframe is defined with the <**iframe**> tag.

## **How to identify the iFrame:**

We cannot detect the frames by just seeing the page or by inspecting Firebug.We can identify the frames in Selenium using methods given below:

* Right click on the element, If you find the option like ‘This Frame’ then it is an iframe.(Please refer the above diagram)
* Right click on the page and click ‘View Page Source’ and Search with the ‘iframe’, if you can find any tag name with the ‘iframe’ then it is meaning to say the page consisting an iframe.

**Case 1: Differtent frames in single web page:**

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
import time  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://www.selenium.dev/selenium/docs/api/java/index.html?overview-summary.html')  
driver.maximize\_window()  
  
driver.**switch\_to.frame**('packageFrame')----using name of frame

packages=driver.find\_element(By.LINK\_TEXT,'org.openqa.selenium').click()  
driver.**switch\_to.default\_content()**----return to driver  
  
driver.**switch\_to.frame**('classFrame') ----using name of frame  
allclasses=driver.find\_element(By.LINK\_TEXT,'AbstractAnnotations').click()  
driver.**switch\_to.default\_content()**-----return to driver

**inner frame**: frame inside a frame

-no need to **switch\_to.default\_content()**

driver.get('https://demo.automationtesting.in/Frames.html')  
driver.maximize\_window()  
  
driver.find\_element(By.XPATH,"//a[normalize-space()='Iframe with in an Iframe']").click()  
  
outer=driver.find\_element(By.XPATH,'/html/body/section/div')-----**outer frame**  
**driver.switch\_to.frame(outer)------using webelement**  
inner=driver.find\_element(By.XPATH,'/html/body/section/div/div')----**inner frame**  
**driver.switch\_to.frame(inner)----using webelement**  
driver.find\_element(By.XPATH,"//input[@type='text']").send\_keys('welcome')

driver.switch\_to.parent\_frame()------**inner frame to outer frame**

**Browser window switch:**

**QUE: 1.**how to switch browser windows depend upon our choice

2.where u will get the window ID

-not present in html it is automatically randomly generated by browser itself at the time of open the new browser

-window id not same for each time its continuously change

-current\_window\_handle ---return window id of single browser window

-window\_handles----return window ids of multiple browser windows

**Switch\_ to.window(‘window id’)**

driver.get('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')----open main window  
driver.maximize\_window()  
  
driver.find\_element(By.LINK\_TEXT,'OrangeHRM, Inc').click()-----Child window open  
windowsIDs=driver.window\_handles  
**method 1:**  
parentwindowid=windowsIDs[0]  
childwindowsid=windowsIDs[1]  
print(parentwindowid,childwindowsid)  
  
driver.switch\_to.window(childwindowsid)-----switch to child window  
print(driver.title)  
  
driver.switch\_to.window(parentwindowid)------switch to parent window  
print(driver.title)

**method 2:**

for winid in windowsIDs:  
 driver.switch\_to.window(winid)  
 print(driver.title)

**close specific window from multiple windows:**

for winid in windowsIDs:  
 driver.switch\_to.window(winid)  
 if driver.title='title of window':  
 driver.close()

**ALERT popup:**

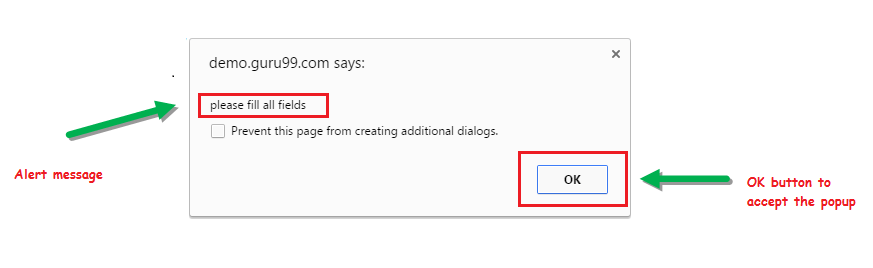
An **Alert in Selenium** is a small message box which appears on screen to give the user some information or notification.

It notifies the user with some specific information or error, asks for permission to perform certain tasks and it also provides warning messages as well.

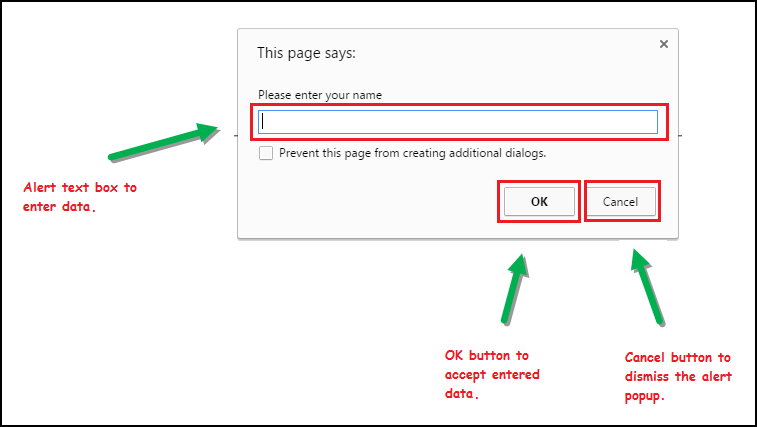
## **Types of Alerts in Selenium**

### 1) Simple Alert

The simple alert class in Selenium displays some information or warning on the screen.

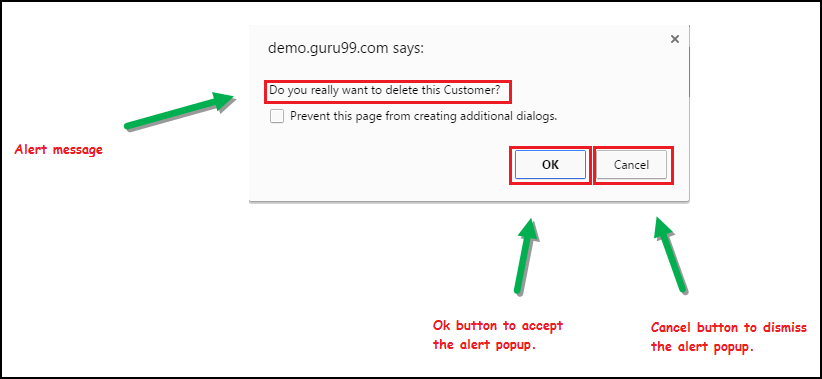
**2) Prompt Alert**

This Prompt Alert asks some input from the user and Selenium webdriver can enter the text using sendkeys(” input…. “).



### 3) Confirmation Alert

This confirmation alert asks permission to do some type of operation.



**How to handle Alert in Selenium WebDriver:**

1) **void dismiss()** // To click on the ‘Cancel’ button of the alert.

**driver.switchTo().alert().dismiss();**

2) **void accept()**// To click on the ‘OK’ button of the alert

**driver.switchTo().alert().accept();**

3) **String getText()** // To capture the alert message.

**driver.switchTo().alert().getText();**

4) **void sendKeys(String stringToSend)** // To send some data to alert box.

**driver.switchTo().alert().sendKeys("Text");**

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
import time  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://rahulshettyacademy.com/AutomationPractice/')  
driver.maximize\_window()  
  
inputbox=driver.find\_element(By.CSS\_SELECTOR,'#name').send\_keys('vr')  
alert=driver.find\_element(By.XPATH,"//input[@id='alertbtn']").click()  
  
alertbox=**driver.switch\_to.alert**  
print(alertbox.text)  
  
alertbox.**accept()**  
*#alertbox.***dismiss()**

Hello vr, share this practice page and share your knowledge----**output**

**Notification Pop UP:**

import time  
from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
  
**ops=webdriver.ChromeOptions()------settings for browser to close thos pop up  
ops.add\_argument('--disable-notifications')**  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
  
driver=webdriver.Chrome(service=service\_obj,**options=ops**)  
  
driver.get('https://www.gps-coordinates.net/my-location')  
driver.maximize\_window()  
  
time.sleep(2)  
driver.close()

**webtable:**

A **Web Table** in Selenium is a WebElement used for the tabular representation of data or information. The data or information displayed can be either static or dynamic. Web table and its elements can be accessed using WebElement functions and locators in Selenium. A typical example of a web table would be product specifications displayed on an eCommerce platform.

**Summary**

* By.xpath() is commonly used to access elements of WebTable in Selenium.
* If the element is written deep within the HTML code such that the number to use for the predicate is very difficult to determine, we can use that element’s unique attribute instead for Selenium get table element.
* Attributes are used as predicates by prefixing them with the @ symbol.
* Use Inspect Element for Accessing WebTable in Selenium

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
import time  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://testautomationpractice.blogspot.com/')  
driver.maximize\_window()

**count no of rows and columns:**

noofrows=driver.find\_elements(By.XPATH,"//table[@name='BookTable']//tr")  
print(len(noofrows))---------7  
  
noofcolumns=driver.find\_elements(By.XPATH,"//table[@name='BookTable']//th")  
print(len(noofcolumns))------4

**read specific row and column data:**

data1=driver.find\_element(By.XPATH,"//table[@name='BookTable'**]//tr[5]/td[1]")**  
print(data1.text)--- Master In Selenium

**read all the rows and columns:**

for r in range(2,len(noofrows)+1):  
 for c in range(1,len(noofcolumns)+1):  
 data1 = driver.find\_element(By.XPATH, "//table[@name='BookTable']//**tr["+str(r)+"]/td["+str(c)+"**]").text  
 print(data1,end=' ')  
 print()

display all table data

**read data based on condition:**

|  |  |  |  |
| --- | --- | --- | --- |
| **BookName** | **Author** | **Subject** | **Price** |
| Learn Selenium | Amit | Selenium | 300 |
| **Learn Java** | **Mukesh** | Java | 500 |
| Learn JS | Animesh | Javascript | 300 |
| **Master In Selenium** | **Mukesh** | Selenium | 3000 |
| Master In Java | Amod | JAVA | 2000 |
| Master In JS | Amit | Javascript | 1000 |

for r in range(2,len(noofrows)+1):  
 authorname = driver.find\_element(By.XPATH,"//table[@name='BookTable']//**tr["+str(r)+"]/td[2]**").tex  
 if authorname == 'mukesh':  
 bookname=driver.find\_element(By.XPATH,"//table[@name='BookTable']//**tr["+str(r)+"]/td[1]"**).text  
 print(bookname,' ', authorname)

**DATE PICKER:**

from selenium import webdriver  
from selenium.webdriver.chrome.service import Service  
from selenium.webdriver.common.by import By  
import time  
  
service\_obj=Service('C:\driver folder\chromedriver\_win32\chromedriver.exe')  
driver=webdriver.Chrome(service=service\_obj)  
  
driver.get('https://jqueryui.com/datepicker/')  
driver.maximize\_window()  
  
driver.switch\_to.frame(0) ---------**switch to frame if having**

**case 1:** direct method to pass date using sendkeys

driver.find\_element(By.XPATH,"//input[@id='datepicker']").send\_keys("05/17/2023")

**case 2:**

y='2024'  
m='september'  
d='29'

driver.find\_element(By.XPATH,"//input[@id='datepicker']").click()---**search window**

while True:  
 month=driver.find\_element(By.XPATH,'//span[@class="ui-datepicker-month"]').text  
 year=driver.find\_element(By.XPATH,'//span[@class="ui-datepicker-year"]').text  
  
 if month==m and y==y:  
 break  
 else:  
 driver.find\_element(By.XPATH,'//\*[@id="ui-datepicker-div"]/div/a[2]/span').click()--------**next key path used for future date if we want past date then used previous key path and proceed**

**first select month and year then go for date always**  
  
date=driver.find\_elements(By.XPATH,'//\*[@id="ui-datepicker-div"]/table/tbody/tr/td/a')---------**select all months date table**  
  
for ele in date:-----------**from date table select specific date**  
 if ele.text==d:  
 ele.click()  
 break