

Selenium

1

Agenda

- Introduction To Selenium
 - What is selenium
 - Components of Selenium
 - Selenium WebDriver
 - Selenium Grid
 - Selenium IDE
 - Why Selenium over other tools
- XPATH
 - Syntax and examples
- CSS Selector
 - Syntax and examples
- Getting Started Selenium-Python
 - Installation
 - Simple usage
 - Example Explained
 - Navigating
 - Interacting with a page
 - Filling in forms
- Locating Elements
 - Locate by ID,Name,Xpath,hyperlinks by link text,TagName,Class Name,CSS Selectors
- Waits
 - Explicit Waits
 - Implicit Waits

Introduction To Selenium

Introduction

- Selenium automates browsers
- It's open source Completely Free , all you need is a system and internet
- Selenium can be executed on multiple platforms
- Can be controlled by many programming languages and testing frameworks
- Support provided from other open-source tools like Jenkins ,ANT...
- Can be integrated with several defect tracking tools

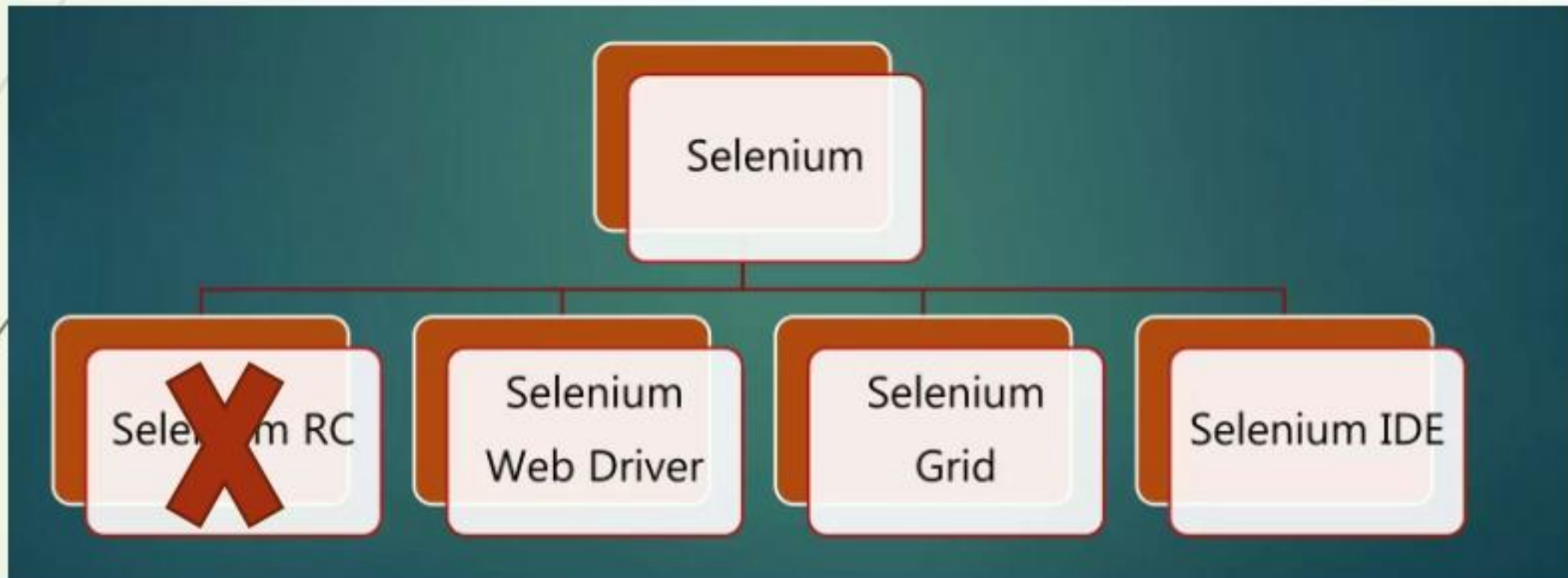
History

- Was developed by Joasin Huggins in 2004. He was working on an in-house project at Thoughtworks. He started working on Selenium Core.
- Dan Fabulich and Nelson Sproul, Created Selenium RC
- Shinya Kasatani in Japan became interested in Selenium and realized that he could wrap the core into an IDE module into the Firefox browser, and be able to record tests as well as play them back in the same plugin

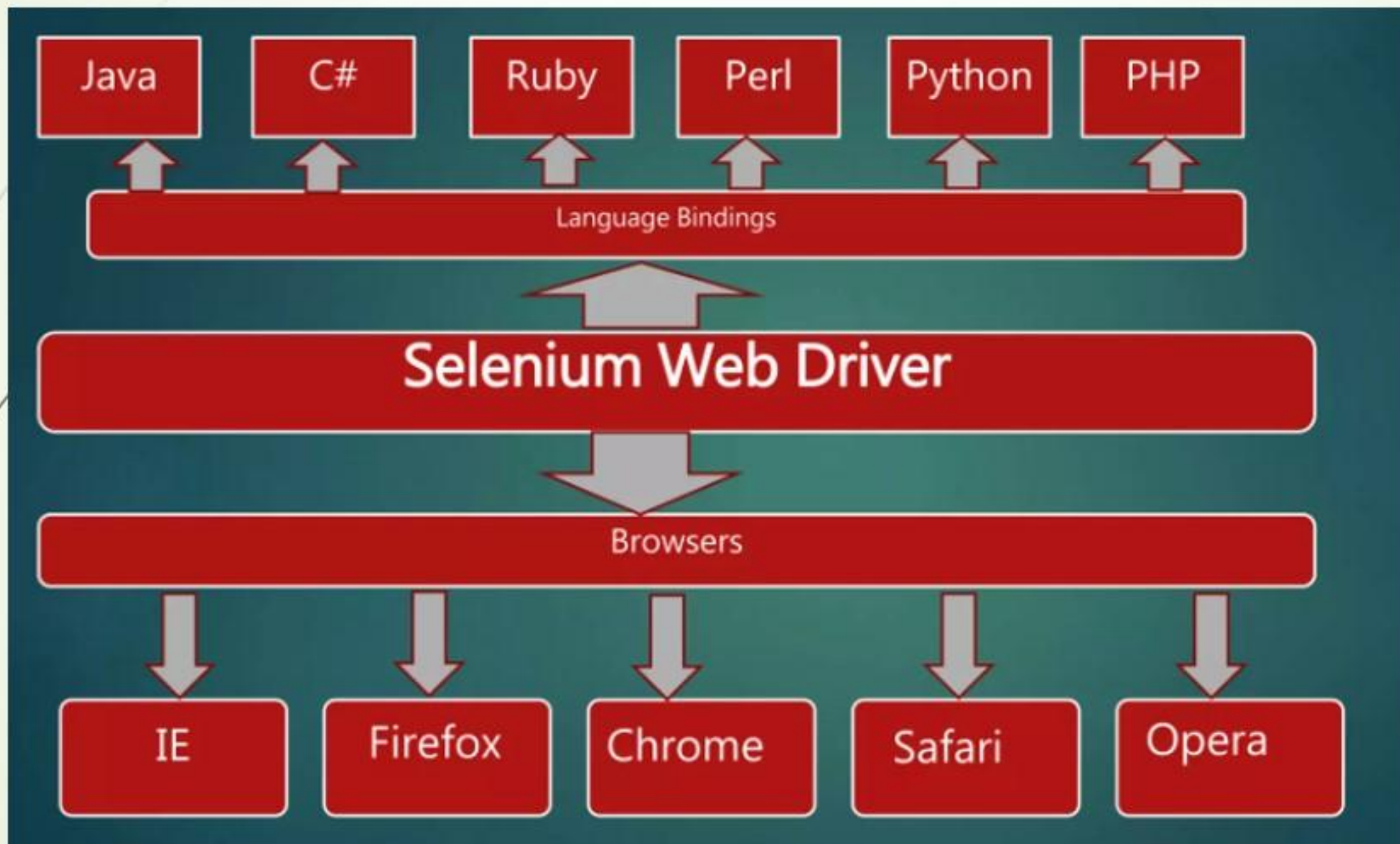
What Is Selenium

- Selenium is a robust tool that supports rapid development of test automation for web-based applications
- Can simulate a user navigating through pages and then assert for specific marks on the pages
- Open-Source web basing testing automation tool and cross-browser compliant
- Open-source tool
- Easy to user

Selenium Components



Selenium Web Driver



Selenium Grid

- Selenium Grid can run large test suites and test that must be run in multiples environments
- Tests can be run in parallel with simultaneous execution (different tests on different remote machines)
- It allows for running your tests in a distributed test execution environment
- Used to run your test against multiple browsers, multiple versions of browser and browser running on different operating systems
- It reduces the time it takes for the test suite to complete a test pass

Features of Selenium IDE

- Easy record and playback
- Intelligent field selection will use Ips , names or Xpath as needed
- Autocomplete for all common selenium commands
- Walk through tests
- Debug and break points
- Save tests as HTML, Ruby scripts or any other format
- Support for Selenium user-extensions.js file
- Option to automatically assert the title of every page

Why Selenium Over other tools?

- Most powerful Open-Source Automation tool available
- Flexible with support to many languages
- Highly extensible
- Platform Support-Provides support on wide range of OS compared to any other tool
- Parallel Testing-Supports parallel testing
- Usability –Easy to use
- ALM integration –Provides integration with several bug tracking tools

XPath

XPath

- XPath is a language used for selecting nodes in an XML document or in an HTML page. It provides a way to navigate through the elements and attributes of an XML or HTML document and access the information they contain

- Syntax

```
# Select element by tag name: //tagname
```

```
Example: //div selects all div elements.
```

```
# Select element by attribute: //*[@attribute='value']
```

```
Example: //*[@id='myid'] selects an element with id attribute equals to myid.
```

```
# Select element by multiple attributes: //*[@attribute1='value1'][@attribute2='value2']
```

```
Example: //input[@type='text'][@name='username'] selects an input element with type equals to text and name equals to username.
```

Xpath(2)

```
# Select element by attribute containing a value: //*[contains(@attribute,'value')]
```

```
Example: //a[contains(@href,'google')] selects all a elements with an href attribute containing the string 'google'.
```

```
# Select element by text: //*[text()='value']
```

```
Example: //h1[text()='Welcome'] selects an h1 element with the text 'Welcome'.
```

```
# Select element by position: (//tagname)[position]
```

```
Example: (//div)[1] selects the first div element.
```

CSS Selector

CSS Selector

- CSS Selector is a pattern used to select and style HTML elements on a web page using Cascading Style Sheets (CSS)

- **Syntax :**

```
# Select an element by ID
```

```
element = driver.find_element(By.CSS_SELECTOR, "#id_element")
```

```
# Select an element by tag name
```

```
element = driver.find_element(By.CSS_SELECTOR, "div")
```

```
# Select an element by class
```

```
element = driver.find_element(By.CSS_SELECTOR, ".ma_classe")
```

```
# Select an element by class using multiple classes
```

```
element = driver.find_element(By.CSS_SELECTOR, ".classe1.classe2")
```

CSS Selector(2)

```
#Select an element by attribute
element = driver.find_element(By.CSS_SELECTOR, "[name='mon_attribut']")

# Select an element by attribute using multiple attributes
element = driver.find_element(By.CSS_SELECTOR, "[name='mon_attribut'][value='ma_valeur']")

# Select an element by pseudo-class
element = driver.find_element(By.CSS_SELECTOR, "a:hover")

# Select an element by position in a list
element = driver.find_element(By.CSS_SELECTOR, "li:nth-of-type(3)")
```


17

Example Html

```
<body>

  <div id="id_element">Je suis un élément avec l'ID "id_element".</div>

  <div>element with type div.</div>

  <div class="ma_classe">element ith class "ma_classe".</div>

  <div class="classe1 classe2">element with "classe1" and "classe2".</div>

  <input type="text" name="mon_attribut" value="ma_valeur">

  <a href="#">Je suis un lien</a>

  <ul>

    <li>Élément 1</li>

    <li>Élément 2</li>

    <li>Élément 3</li>

    <li>Élément 4</li>

  </ul>

</body>
```

```
# ELEMENT                                     ==> CSS SELECTOR

# div id="id_element" ==> #id_element

# div ==> div

# div class="ma_classe" ==> .ma_classe

# div class="classe1 classe2" ==> .classe1.classe2

# input type="text" name="mon_attribut" value="ma_valeur"
==> [name='mon_attribut'][value='ma_valeur']

# a href="#" ==> a:hover

# ul li:nth-of-type(3) ==> li:nth-of-type(3)
```


Getting Started with Selenium-Python

Simple usage

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
driver = webdriver.Chrome()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element(By.NAME, "q")
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```

Explained Example

```
from selenium import webdriver
```

- selenium.webdriver module provides all the WebDriver implementations

```
from selenium.webdriver.common.by import keys
```

- The Keys class provide keys in the keyboard like RETURN, F1, ALT etc.

```
from selenium.webdriver.common.keys import By
```

- The By class is used to locate elements within a document.

Explained Example

```
driver = webdriver.Chrome()
```

- The instance of Chrome Webdriver is created

```
driver.get("http://www.python.org")
```

- The driver.get method will navigate to a page given by the URL. WebDriver will wait until the page has fully loaded (that is, the “onload” event has fired) before returning control to your test or script

```
assert "Python" in driver.title
```

- assertion to confirm that title has the word “Python”

```
elem = driver.find_element(By.NAME, "q")
```

- WebDriver offers a number of ways to find elements using the find_element method. For example, the input text element can be located by its name attribute using the find_element method and using By.NAME as its first parameter

Explained Example

```
elem.clear()
```

```
elem.send_keys("pycon")
```

```
elem.send_keys(Keys.RETURN)
```

- ▶ we are sending keys, this is similar to entering keys using your keyboard. Special keys can be sent using the Keys class imported from selenium.webdriver.common.keys. To be safe, we'll first clear any pre-populated text in the input field

```
assert "No results found." not in driver.page_source
```

- ▶ After submission of the page, you should get the result if there is any. To ensure that some results are found, make an assertion

```
driver.close()
```

- ▶ Finally, the browser window is closed. You can also call the quit method instead of close. The quit method will exit the browser whereas close will close one tab, but if just one tab was open, by default most browsers will exit entirely

Navigating

Interacting with the page

- For example, given an element defined as:

```
<input type="text" name="passwd" id="passwd-id" />
```

- you could find it using any of:

```
element = driver.find_element(By.ID, "passwd-id")
element = driver.find_element(By.NAME, "passwd")
element = driver.find_element(By.XPATH, "//input[@id='passwd-id']")
element = driver.find_element(By.CSS_SELECTOR, "input#passwd-id")
```

- You may want to enter some text into a text field

```
element.send_keys("some text")
```

- You can simulate pressing the arrow keys by using the “Keys” class:

```
element.send_keys(" and some", Keys.ARROW_DOWN)
```

Filling in forms

- We've already seen how to enter text into a textarea or text field, but what about the other elements? You can "toggle" the state of the drop down, and you can use "setSelected" to set something like an *OPTION* tag selected.

```
element = driver.find_element(By.XPATH, "//select[@name='name']")
all_options = element.find_elements(By.TAG_NAME, "option")
for option in all_options:
    print("Value is: %s" % option.get_attribute("value"))
    option.click()
```

```
<!-- Html element -->
<select name="name">
    <option value="option1">Option 1</option>
    <option value="option2">Option 2</option>
    <option value="option3">Option 3</option>
</select>
```


Filling in forms (2)

- WebDriver's support classes include one called a "Select", which provides useful methods for interacting with these forms

```
from selenium.webdriver.support.ui import Select  
  
select = Select(driver.find_element(By.NAME, 'name'))  
  
select.select_by_index(index)  
  
select.select_by_visible_text("text")  
  
select.select_by_value(value)
```

- WebDriver also provides features for deselecting all the selected options:

```
select = Select(driver.find_element(By.ID, 'id'))  
select.deselect_all()
```

```
<!-- Html element -->  
<select name="langue">  
    <option value="fr">Français</option>  
    <option value="en">Anglais</option>  
    <option value="es">Espagnol</option>  
</select>
```


Filling in forms (3)

- Suppose in a test, we need the list of all default selected options, Select class provides a property method that returns a list

```
select = Select(driver.find_element(By.XPATH, "//select[@name='name']"))  
all_selected_options = select.all_selected_options
```

- To get all available options:

```
options = select.options
```

- To submit the form :

```
➤ # Assume the button has the ID "submit" :)  
➤ driver.find_element_by_id("submit").click()
```

Locating Elements

Locating Elements

- Selenium provides the following method to locate elements in a page:

```
driver.find_element(By.NAME, "username") # to find one element  
driver.find_elements(By.NAME, "image") # to find multiple elements
```

- Example usage

```
from selenium.webdriver.common.by import By  
driver.find_element(By.XPATH, '//button[text()='Some text']')  
driver.find_elements(By.XPATH, '//button')
```

- The attributes available for the *By* class are used to locate elements on a page. These are the attributes available for *By* class:


```
➤ driver.find_element(By.ID, "id")  
➤ driver.find_element(By.NAME, "name")  
➤ driver.find_element(By.XPATH, "xpath")  
➤ driver.find_element(By.LINK_TEXT, "link text")  
➤ driver.find_element(By.PARTIAL_LINK_TEXT, "partial link text")  
➤ driver.find_element(By.TAG_NAME, "tag name")  
➤ driver.find_element(By.CLASS_NAME, "class name")  
➤ driver.find_element(By.CSS_SELECTOR, "css selector")
```


Locating by Id

- Html page source

```
<html>
<body>
  <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
  </form>
</body>
</html>
```

- The form element can be located like this:




```
login_form = driver.find_element(By.ID, 'loginForm')
```

Locating by Name

➤ Html page source


```
<html>
<body>
  <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```

- The username & password elements can be located like this:



```
username = driver.find_element(By.NAME, 'username')
password = driver.find_element(By.NAME, 'password')
```

- This will give the “Login” button as it occurs before the “Clear” button:




```
login = driver.find_element(By.NAME, 'continue')
```


Locating by XPath

- Html page source

```
<html>
<body>
  <form id="loginForm">
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```

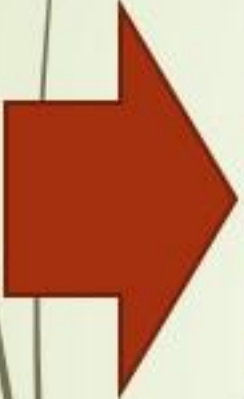
- The form elements can be located like this:



```
➤ login_form = driver.find_element(By.XPATH, "/html/body/form[1]") # Absolute path
➤ login_form = driver.find_element(By.XPATH, "//form[1]") #First form element in the HTML
➤ login_form = driver.find_element(By.XPATH, "//form[@id='loginForm']") # The form element with
attribute id set to loginForm
```


Locating by Xpath (2)

- The username element can be located like this:




```
username = driver.find_element(By.XPATH, "//form[input/@name='username']")
#First form element with an input child element with name set to username

username = driver.find_element(By.XPATH, "//form[@id='loginForm']/input[1]")
#First input child element of the form element with attribute id set to loginForm

username = driver.find_element(By.XPATH, "//input[@name='username']")
#First input element with attribute name set to username
```

- The “Clear” button element can be located like this:



```
clear_button = driver.find_element(By.XPATH, "//input[@name='continue'][@type='button']")
#Input with attribute name set to continue and attribute type set to button


clear_button = driver.find_element(By.XPATH, "//form[@id='loginForm']/input[4]")
#Fourth input child element of the form element with attribute id set to loginForm
```

Locating Hyperlinks by Link Text

- ▶ Html page source

```
<html>
  <body>
    <p>Are you sure you want to do this?</p>
    <a href="continue.html">Continue</a>
    <a href="cancel.html">Cancel</a>
  </body>
</html>
```

- ▶ The continue.html link can be located like this:



```
continue_link = driver.find_element(By.LINK_TEXT, 'Continue')
continue_link = driver.find_element(By.PARTIAL_LINK_TEXT, 'Conti')
```


Locating by Tag Name

- ▶ Html page source

```
<html>  
  
  <body>  
  
    <h1>Welcome</h1>  
  
    <p>Site content goes here.</p>  
  
  </body>  
  
</html>
```

- ▶ The heading (h1) element can be located like this:



```
heading1 = driver.find_element(By.TAG_NAME, 'h1')
```


Locating by Class Name

- ▶ Html page source

```
<html>  
  <body>  
    <p class="content">Site content goes here.</p>  
  </body>  
</html>
```

- ▶ The “p” element can be located like this:



```
content = driver.find_element(By.CLASS_NAME, 'content')
```

Locating by CSS Selector

- ▶ Html page source

```
<html>
  <body>
    <p class="content">Site content goes here.</p>
  </body>
</html>
```

- ▶ The “p” element can be located like this:



```
content = driver.find_element(By.CSS_SELECTOR, 'p.content')
```


Waits

Explicit Waits

➤ Code source


```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("http://somedomain/url_that_delays_loading")
element = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "myDynamicElement")))
driver.quit()
```

- 
- Selenium will wait for a maximum of 10 seconds for an element matching the given criteria to be found.
 - If no element is found in that time, a `TimeoutException` is thrown
 - By default, `WebDriverWait` calls the `ExpectedCondition` every 500 milliseconds until it returns success
 - `ExpectedCondition` will return *true* (Boolean) in case of success or *not null* if it fails to locate an elemen

Explicit Waits(Expected Conditions)

- There are some common conditions that are frequently of use when automating web browser



```
EC.title_is
```

```
EC.title_contains
```

```
EC.presence_of_element_located
```

```
EC.visibility_of_element_located
```

```
EC.visibility_of
```

```
EC.presence_of_all_elements_located
```

```
EC.text_to_be_present_in_element
```

```
EC.text_to_be_present_in_element_value
```

```
EC.frame_to_be_available_and_switch_to_it
```

```
EC.invisibility_of_element_located
```

```
EC.element_to_be_clickable
```

```
EC.staleness_of
```

```
EC.element_to_be_selected
```

```
EC.element_located_to_be_selected
```

```
EC.element_selection_state_to_be
```

```
EC.element_located_selection_state_to_be
```

```
EC.alert_is_present
```


Implicit Waits

- An implicit wait tells WebDriver to poll the DOM for a certain amount of time when trying to find any element (or elements) not immediately available
- The default setting is 0 (zero)
- Once set, the implicit wait is set for the life of the WebDriver object.
- Example source code

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.implicitly_wait(10) # seconds

driver.get("http://somedomain/url_that_delays_loading")

myDynamicElement = driver.find_element_by_id("myDynamicElement")
```