Your Selenium Code

|

▼

WebDriver API (JSON Wire / W3C Protocol)

|

▼

Browser Driver (chromedriver / geckodriver)

|

▼

Browser (Chrome, Firefox, etc.)

Basic Locators in Selenium

1. **id** – Fastest and most preferred
2. **name**
3. **class_name**
4. **tag_name**
5. **link_text** / **partial_link_text**
6. **xpath** – Flexible but slower
7. **css_selector** – Flexible and faster than xpath

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

# Create Chrome browser instance
driver = webdriver.Chrome()  # If chromedriver is in PATH

# Maximize window
driver.maximize_window()

# Go to login page
driver.get("https://practicetestautomation.com/practice-test-login/")

# Find username field and type text
driver.find_element(By.ID, "username").send_keys("student")

# Find password field and type text
driver.find_element(By.ID, "password").send_keys("Password123")

# Click login button
driver.find_element(By.ID, "submit").click()
```

```
# Wait to see result
time.sleep(3)

# Close browser
driver.quit()
```

parent:nth-child(index)
parent:nth-of-type(index)

XPath	CSS Equivalent
`//input[@id='username']`	`input#username`
`//input[@name='username']`	`input[name='username']`
`//*[@class='form-control']`	`.form-control`
`//div[@id='main']//a`	`div#main a`
`//form[@id='login']/input[1]`	`form#login > input:nth-of-type(1)`

A. Implicit Wait

- **Global** for the whole `driver` session.
- Selenium will try to find the element until the timeout expires.
- Once set, applies to **all `find_element` calls**.
- Polling happens every 500 ms.

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

driver = webdriver.Chrome()
driver.implicitly_wait(10)  # wait max 10 seconds for elements

driver.get("https://example.com")
element = driver.find_element(By.ID, "username")
```

Explicit Wait

- Waits for a **specific element** and **specific condition**.
- Uses `WebDriverWait` + `expected_conditions`.
- **Best choice** for flaky tests.
- Polls every 500 ms (can be changed in fluent wait).

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, 10)  # wait max 10 sec

username = wait.until(EC.visibility_of_element_located((By.ID, "username")))
username.send_keys("admin")
```

W**hen to use:**

- When you need **different waits for different elements**.
- When page load times vary.
- For dynamic content like AJAX, animations.

Fluent Wait

- Same as explicit wait but **you control polling frequency** and **ignore exceptions**.
- Useful for highly dynamic apps with unstable element timings.

```
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import NoSuchElementException

wait = WebDriverWait(driver, timeout=15, poll_frequency=2,
ignored_exceptions=[NoSuchElementException])

element = wait.until(EC.presence_of_element_located((By.ID, "username")))
element.send_keys("admin")
```

Scenario	Best Wait
Page loads slow but consistently	**Implicit Wait**
Element appears at random time	**Explicit Wait**
Need custom polling & ignore exceptions	**Fluent Wait**
Most real-world automation	**Mix of Implicit + Explicit** (but avoid using both heavily in same test — can cause confusion)

```
# Switch to alert
alert = driver.switch_to.alert
alert.accept()     # OK
# alert.dismiss()  # Cancel (for confirmation alert)
# alert.send_keys("Hello")  # For prompt alert

windows = driver.window_handles
driver.switch_to.window()

switch_to.frame()  switch_to.default_content() switch_to.parent_frame()
```

Scrolling in Selenium

Selenium can scroll in two main ways:

- **Via JavaScript execution** (`execute_script`)
- **Via WebElement actions** (`scrollIntoView`)

a) Scroll by pixel amount

```
driver.execute_script("window.scrollBy(0, 500);")  # down 500px
driver.execute_script("window.scrollBy(0, -500);") # up 500px
```

b) Scroll to bottom of page

```
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
```

Scroll to element

```
element = driver.find_element(By.ID, "myElement")
driver.execute_script("arguments[0].scrollIntoView(true);", element)
```

Executing JavaScript in Selenium
title = driver.execute_script("return document.title;")
print(title)

b) Click an element via JS

```
element = driver.find_element(By.ID, "submitBtn")
driver.execute_script("arguments[0].click();", element)
```

> Useful when normal `.click()` fails due to overlay or hidden element.

c) Send keys (set value) via JS

```
element = driver.find_element(By.ID, "username")
driver.execute_script("arguments[0].value = 'myUsername';", element)
```

---

d) Get text from an element via JS

```
text = driver.execute_script("return arguments[0].textContent;", element)
```

2️⃣ Using the Select Class

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import Select
import time

driver = webdriver.Chrome()
driver.get("https://www.example.com")

# Locate the dropdown
dropdown_element = driver.find_element(By.ID, "cars")

# Create Select object
select = Select(dropdown_element)

# 1. Select by visible text
select.select_by_visible_text("BMW")

# 2. Select by index (0-based)
select.select_by_index(2)  # Selects Audi
```

```
# 3. Select by value attribute
select.select_by_value("volvo")

time.sleep(2)
driver.quit()
```

Broken links:

```
# Get all links
links = driver.find_elements("tag name", "a")

for link in links:
    url = link.get_attribute("href")
    if url is None or url.startswith("javascript"):
        continue  # Skip empty or JS links

    try:
        response = requests.head(url, allow_redirects=True)  # Only fetch headers for speed
        if response.status_code >= 400:
            print(f"❌ Broken link: {url} (Status: {response.status_code})")
        else:
            print(f"✅ Valid link: {url}")
    except requests.exceptions.RequestException as e:
        print(f"⚠ Error checking {url}: {e}")
```

Always **skip**:

- Links with `javascript:void(0)`
- Empty `href`

Actions Commands (Interaction)

These commands **perform actions** on web elements or the browser.

Examples:

- **Clicking:**

  ```
  element.click()
  ```

- **Typing text:**

  ```
  element.send_keys("username123")
  ```

- **Clearing a field:**

  ```
  element.clear()
  ```

- **Navigating to a URL:**

  ```
  driver.get("https://example.com")
  driver.back()
  driver.forward()
  driver.refresh()
  ```

2️⃣ Accessor Commands (Get/Read Data)

These commands **retrieve information** about the page or elements.

Examples:

- **Get page title:**

  ```
  driver.title
  ```

- **Get current URL:**

  ```
  driver.current_url
  ```

- **Get element text:**

  ```
  element.text
  ```

- **Get attribute value:**

  ```
  element.get_attribute("href")
  ```

- **Get size/position:**

  ```
  element.size
  element.location
  ```