

Introduction to locators

Introduction to Locators

Key concepts for finding and interacting with elements in web automation frameworks (e.g., Selenium, Playwright, Cypress).

1. What Is a Locator?

- **Definition:** A locator is a selector that uniquely identifies a UI element on a web page (button, link, input, etc.) so that automation scripts can interact with it.
 - **Purpose:** Enables reliable, maintainable test scripts; abstracts the underlying HTML structure.
-

2. Why Choose the Right Locator?

Reason	Impact of a Poor Locator
Stability	Fragile locators break with minor UI changes.
Readability	Clear locators make scripts self-documenting.
Performance	Efficient selectors reduce lookup time.
Maintainability	Centralized locators simplify updates.

3. Locator Strategies (Common to Most Frameworks)

Strategy	Syntax (Selenium/Python)	Typical Use Case	Pros	Cons
ID	<code>driver.find_element(By.ID, "username")</code>	Unique element IDs	Fast, unique	Requires developer-provided IDs
Name	<code>driver.find_element(By.NAME, "email")</code>	Form fields	Simple	May not be unique

Introduction to locators

Strategy	Syntax (Selenium?Python)	TypicalUse ?Case	Pros	Cons
Class Name	<code>driver.find_element(By.CLASS_NAME, "btn-primary")</code>	Styling hooks	Easy	Often shared by many elements
Tag Name	<code>driver.find_element(By.TAG_NAME, "button")</code>	Generic tag search	Useful for collections	Too broad alone
Link Text / Partial Link Text	<code>driver.find_element(By.LINK_TEXT, "Forgot Password")</code>	Anchor links	Human? readable	Sensitive to text changes
CSS Selector	<code>driver.find_element(By.CSS_SELECTOR, "input[type='email']")</code>	Any CSS pattern	Very flexible, fast	Complex syntax can be hard to read
XPath	<code>driver.find_element(By.XPATH, "//div[@class='error']/span")</code>	Hierarchical navigation	Powerful, can locate any element	Slower, brittle if structure changes
Custom Attribute	<code>driver.find_element(By.CSS_SELECTOR, "[data-test='login-btn']")</code>	Test? specific attributes	Stable, independent of UI	Requires dev cooperation

4. Best Practices for Choosing Locators

1. **Prefer IDs** – they are unique and fastest.
2. **Use meaningful custom attributes** (data-test, data-qa, data-cy) when IDs are unavailable.
3. **Avoid absolute XPaths** – they depend on the full DOM hierarchy. Use relative XPaths instead.

Introduction to locators

4. **Keep selectors short but specific** – e.g., `button.submit` is better than `div > form > button`.
 5. **Leverage CSS over XPath** when possible (generally faster and more readable).
 6. **Encapsulate locators** in a dedicated object?repository or page?object class.
 7. **Document intent** – add comments or descriptive variable names (`loginButton`, `errorMessageSpan`).
-

5. Page Object Model (POM) – Organizing Locators

```
# example: login_page.py
from selenium.webdriver.common.by import By

class LoginPage:
    # Locators
    USERNAME_INPUT = (By.ID, "username")
    PASSWORD_INPUT = (By.ID, "password")
    SUBMIT_BTN     = (By.CSS_SELECTOR, "button[type='submit']")
    ERROR_MSG      = (By.XPATH, "//div[@class='alert']/p")

    # Actions
    def
```