XPath & CSS Selectors: Mastering Locator Techniques in Test Automation

(Automation QA/SDET)

Prepared by:-

Kushal Parikh

1) XPath Techniques for Custom Locators

XPath is a powerful XML-based query language used to traverse and locate elements within the DOM.

- **Using `contains()` for Partial Matching**

`contains()` is useful when attributes such as IDs or class names are dynamic.

```
//input[contains(@id, 'username')]
```

This will locate any `<input>` element where the `id` attribute contains "username".

1) Using `starts-with()` for Matching Prefixes

This function is helpful when an attribute starts with a known prefix but changes dynamically.

```
//button[starts-with(@id, 'btn_')]
```

This will locate any `<button>` element whose `id` begins with "btn_".

2) Using `text()` to Locate Elements by Visible Text

For elements with static text values, `text()` can be used for direct matching.

```
//button[text()='Submit']
```

This will locate any `<button>` element containing the exact text "Submit".

3) Using `normalize-space()` to Handle Extra Spaces

If an element contains leading or trailing spaces, `normalize-space()` ensures correct identification.

```
//button[normalize-space()='Proceed']
```

This will locate `<button>  Proceed  </button>` even if unnecessary spaces exist.

4) Combining Multiple Conditions Using `OR` and `AND` Operators

- **Using `OR` Operator**

Selects elements that match either condition.

```
//input[@type='submit' or @type='button']
```

This will select an `<input>` element where the `type` is either "submit" or "button".

- **Using `AND` Operator**

Selects elements that satisfy both conditions.

```
//input[@type='text' and @name='username']
```

This will select an `<input>` element where the `type` is "text" and the `name` is "username".

5) XPath Axes for Complex Element Navigation

- **Parent Selection**

```
//span[text()='Username']/parent::label
```

Finds the `<label>` parent of a `<span>` containing "Username".

- **Child Selection**

```
//div[@class='container']/child::ul
```

Finds the `<ul>` child of a `<div>` with class "container".

- **Following Sibling Selection**

```
//label[text()='Password']/following-sibling::input
```

Finds an `<input>` element that follows a `<label>` containing "Password".

- **Preceding Sibling Selection**

```
//input[@id='password']/preceding-sibling::label
```

Finds the `<label>` that appears before an `<input>` with `id="password"`.

- **Ancestor Selection**

```
//input[@id='search']/ancestor::form
```

Finds the closest `<form>` ancestor of an `<input>` with `id="search"`.

- **Descendant Selection**

```
//div[@class='wrapper']/descendant::a
```

Finds all `<a>` elements nested within a `<div>` with class "wrapper".

6) XPath `translate()` Function

The `translate()` function in XPath replaces or removes specific characters in a string. It is useful for case-insensitive matching, removing unwanted characters, and normalizing text in test automation.

Syntax:

```
translate(source-string, characters-to-replace, replacement-characters)
```

Common Use Cases

✓ Convert to Lowercase (Case-Insensitive Matching)

```
//input[translate(@name, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'abcdefghijklmnopqrstuvwxyz')='username']
```

✓ Remove Spaces, Dashes, or Special Characters

```
//phone[translate(text(), '- ', '')='1234567890']
```

✓ Remove Currency Symbols or Commas from Numbers

```
//price[translate(text(), '$,', '')='1000']
```

✓ Remove All Digits from Text

```
//div[translate(text(), '0123456789', '')='Total Amount']
```

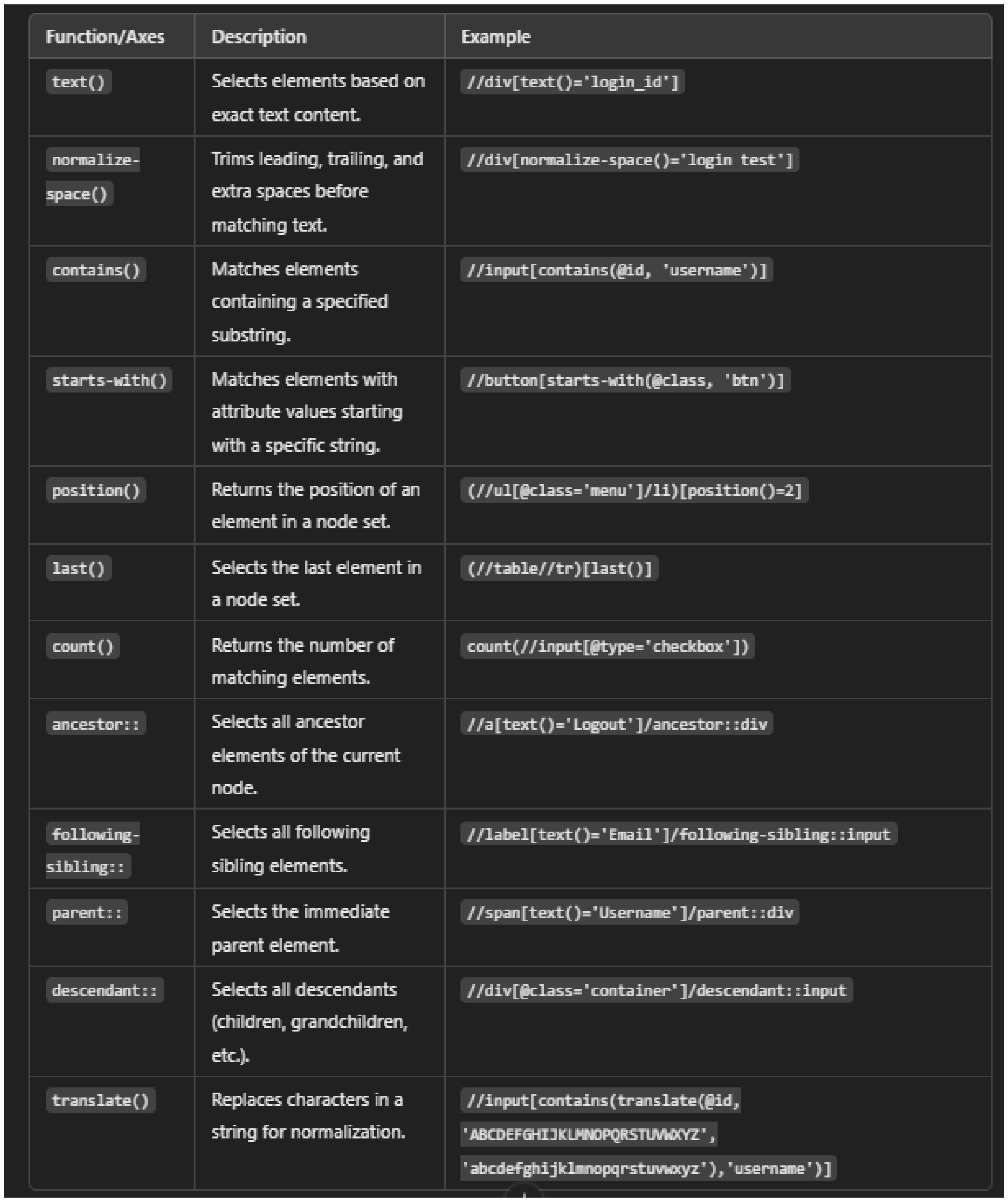
- **Limitations**
  - ➢ No regex support (cannot replace substrings).
  - ➢ Character-to-character replacement only (must match lengths).

- **Best Uses in Test Automation**
  - ➢ Case-insensitive element selection
  - ➢ Cleaning dynamic text values
  - ➢ Removing unnecessary symbols for precise matching

`translate()` helps create robust XPath locators by ensuring consistency in text-based searches.

Checkout my Topmate : https://topmate.io/kushalparikh11/

Quick XPath Cheat Sheet



Function/Axes	Description	Example
`text()`	Selects elements based on exact text content.	`//div[text()='login_id']`
`normalize-space()`	Trims leading, trailing, and extra spaces before matching text.	`//div[normalize-space()='login test']`
`contains()`	Matches elements containing a specified substring.	`//input[contains(@id, 'username')]`
`starts-with()`	Matches elements with attribute values starting with a specific string.	`//button[starts-with(@class, 'btn')]`
`position()`	Returns the position of an element in a node set.	`(//ul[@class='menu']/li)[position()=2]`
`last()`	Selects the last element in a node set.	`(//table//tr)[last()]`
`count()`	Returns the number of matching elements.	`count(//input[@type='checkbox'])`
`ancestor::`	Selects all ancestor elements of the current node.	`//a[text()='Logout']/ancestor::div`
`following-sibling::`	Selects all following sibling elements.	`//label[text()='Email']/following-sibling::input`
`parent::`	Selects the immediate parent element.	`//span[text()='Username']/parent::div`
`descendant::`	Selects all descendants (children, grandchildren, etc.).	`//div[@class='container']/descendant::input`
`translate()`	Replaces characters in a string for normalization.	`//input[contains(translate(@id, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz'),'username')]`

2) CSS Selector Techniques for Custom Locators

CSS Selectors provide a more performance-optimized and concise way of identifying elements compared to XPath.

Follow me on LinkedIn : https://www.linkedin.com/in/kushalparikh11/

1) Basic CSS Selectors

- **By Tag Name**

```
div
```

Selects all `<div>` elements.

- **By ID (``)**

```
#login-button
```

Selects an element with `id="login-button"`.

- **By Class (`.`)**

```
.btn-primary
```

Selects all elements with class `btn-primary`.

- **By Attribute**

```
input[type='text']
```

Selects `<input>` elements with `type="text"`.

2) Advanced CSS Selectors

- **Direct Child (`>`)**

Selects only direct child elements.

```
div > p
```

Finds `<p>` elements that are direct children of a `<div>`.

- **Descendant (`space`)**

Selects any nested child element.

```
div p
```

Finds `<p>` elements inside `<div>`, regardless of nesting depth.

- **Adjacent Sibling (`+`)**

Selects the next sibling element.

```
h2 + p
```

Finds the first `<p>` element immediately after an `<h2>`.

- **General Sibling (`~`)**

Selects all matching siblings.

```
h2 ~ p
```

Finds all `<p>` elements that are siblings of an `<h2>`.

---

3) Using `nth-child()` and `nth-of-type()` for Indexed Elements

- **Using `nth-child()`**

```
ul li:nth-child(3)
```

Finds the third `<li>` inside a `<ul>`.

- **Using `nth-of-type()`**

```
div:nth-of-type(2)
```

Finds the second `<div>` inside a parent, regardless of other elements.

---

4) Attribute Wildcard Matching

- **Contains (`*`)**

```
input[name*='user']
```

Finds `<input>` elements where the `name` contains "user".

- **Starts With (`^`)**

```
input[name^='first']
```

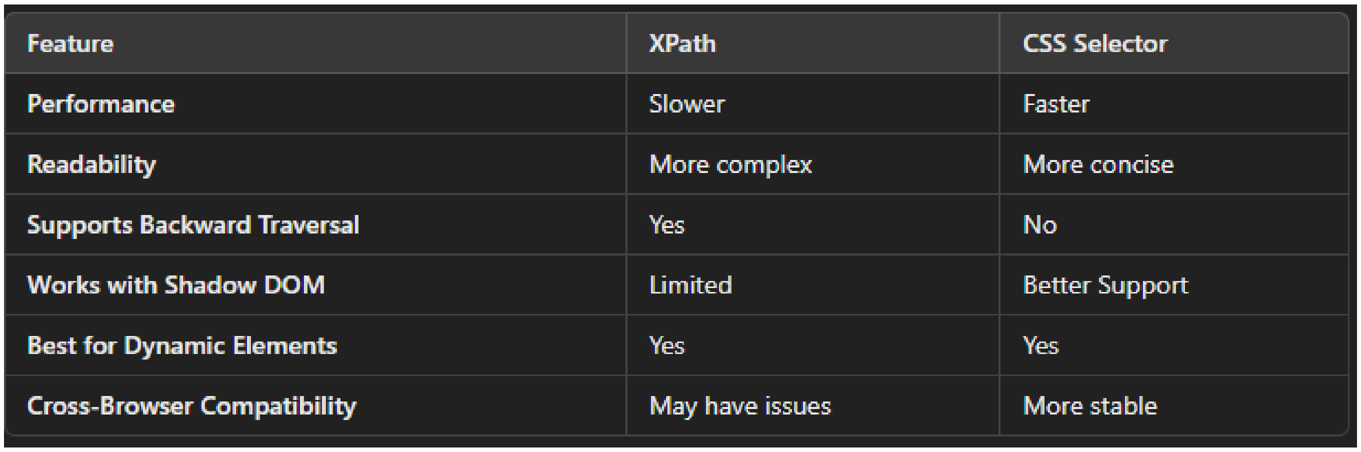
Finds `<input>` elements where the `name` starts with "first".

- **Ends With (`$`)**

```
input[name$='name']
```

Finds `<input>` elements where the `name` ends with "name".

3) Choosing Between XPath and CSS Selectors



Feature	XPath	CSS Selector
Performance	Slower	Faster
Readability	More complex	More concise
Supports Backward Traversal	Yes	No
Works with Shadow DOM	Limited	Better Support
Best for Dynamic Elements	Yes	Yes
Cross-Browser Compatibility	May have issues	More stable

CSS Selectors should be preferred for performance reasons, but XPath is necessary when navigating backward in the DOM or when CSS Selectors are insufficient for complex structures.

4) Best Practices for Writing Robust Locators

- Use Unique IDs and Data Attributes whenever available.
- Avoid absolute XPath (`/html/body/...`) as it is fragile.
- Use CSS Selectors for speed and XPath for complex parent-child relationships.
- Leverage XPath functions like `contains()`, `starts-with()`, and `text()` for handling dynamic elements.
- Always test locators in browser DevTools (`F12` → Elements tab) before using them in automation scripts.