XPath for

Test Automation

Interview Questions and Answers & Solved Quiz Questions

*by Inder P Singh for **SDETs, QA** and **Testers***

Download for reference ⬇️ Like 👍 Share 🔗

YouTube Channel: Software and Testing Training (341 Tutorials, 9.9 Million Views, 82.1 K Subscribers)

Blog: Software Testing Space (484 Posts, 1.82 Million Views)

1 What is XPath and XPath Syntax Basics

Question: What is **XPath**? Why is it important in test automation?

Answer: XPath stands for XML Path Language. It is a powerful query language for locating nodes within an XML or HTML document. View XPath introduction in the first XPath tutorial here. In test automation, especially with tools like Selenium WebDriver (view How to find XPath in Selenium WebDriver tutorials here and here and view Selenium Java interview questions and answers full set here), **XPath** is used to locate web page elements (such as buttons, links, fields) by navigating the document's DOM tree structure. XPath allows testers to find elements based on various criteria (tags, attributes, text, relationships, etc.), because XPath is flexible. It can locate elements even if they lack an ID or name, by creating expressions that uniquely identify them.

Example: For example, to find an element with an ID attribute `searchBox`, one can use:

```
driver.findElement(By.xpath("//*[@id='searchBox']"))
```

Question: What is the difference between absolute and relative **XPath**?

Answer: An **absolute XPath** starts from the root node of the document and follows the full hierarchy of elements down to the target node. For instance, `/html/body/div[2]/div[@class='header']/span` specifies the exact path from the root. Absolute XPaths are very specific but tend to be brittle, because any change in the page structure (like adding a new container such as a `div`) can break them.

A **relative XPath**, on the other hand, starts anywhere in the document using `//` and matches nodes at any level by attribute or tag. For example, `//input[@type='text']` finds any text input field without relying on its full path. Relative XPaths are generally preferred for maintainability purpose.

Example:

```
String absoluteXPath = "/html/body/div[2]/div[@class='header']/span";

String relativeXPath = "//div[@class='header']/span";
```

Question: What are nodes, predicates and operators in **XPath** syntax?

Answer: In **XPath**, a *node* usually refers to an element in the DOM, such as `div`, `span`, or `@class`. An XPath expression is a path through these nodes. A *predicate* is a condition placed in square brackets `[...]` that filters the selected nodes. For example, in `//input[@type='text']`, `[@type='text']` is a predicate filtering `input` elements by their `type` attribute. *Operators* like `and`, `or`, `=`, `!=`, etc., can be used inside predicates to

combine conditions. **XPath** also provides functions like `contains()` and `starts-with()` for refining matches. Nodes, predicates, and operators let us precisely select elements that match complex criteria.

Example: To select a list item only if it has a specific class and contains certain text, you might write:

```
driver.findElement(By.xpath("//li[@class='item' and
contains(text(),'Software and Testing Training')]"))
```

Quiz:

1. Which starting symbol denotes a relative XPath?
   A. /
   B. // (Correct)
   C. .
   D. ..
2. What does a *predicate* in an XPath expression represent?
   A. A function call
   B. A node name
   C. A condition filter (Correct)
   D. An XPath axis
3. Which operator would you use inside a predicate to require multiple conditions?
   A. +
   B. and (Correct)
   C. |
   D. =
4. What is the main disadvantage of using an absolute XPath?
   A. It's slower to execute
   B. It only works in Internet Explorer
   C. It breaks easily if the page structure changes (Correct)
   D. It cannot select elements by attributes

2 Basic XPath Expressions

Question: How can you select elements by **tag name** using XPath?
Answer: To select elements by tag name, use `//tagName`. This selects all matching tags in the DOM. For example, `//div` matches all `<div>` elements. Tag-based XPath is typically used to find elements when other attributes aren't available.
Example: To find all list item elements, you could write:

```
driver.findElements(By.xpath("//li"));
```

Question: How do you locate elements using **attributes** in XPath?
Answer: XPath allows selecting elements by attributes using the @ symbol. The syntax `//tag[@attribute='value']` finds elements with a given attribute value. Using * can match any tag with that attribute. For instance, `//*[@class='button']` will find any element with `class="button"`. This is useful when the tag type can vary or is not known.
Example: To find an input field with name attribute 'username':

```
driver.findElement(By.xpath("//input[@name='username']"));
```

Question: How is the **text()** function used in XPath expressions?
Answer: The `text()` function lets XPath match elements based on their visible text content. By writing `//tag[text()='ExactText']`, you can find an element whose inner text is exactly 'ExactText'. This is useful for buttons or labels where the text is unique. However, exact text matching is case-sensitive and must match the full text node.
Example: To find a button labeled 'Submit':

```
driver.findElement(By.xpath("//button[text()='Submit']"));
```

```
// © Software Testing Space https://inderpsingh.blogspot.com/
```

Question: What does the **contains()** function do in XPath?
Answer: The `contains()` function checks if a node's attribute or text contains a given substring. Using syntax like `contains(text(),'subText')` or `contains(@class,'partial')`, you can match partial values. This is very helpful for dynamic content or when the full text/attribute value is unpredictable. For instance, `//label[contains(text(),'User')]` would match a `<label>` containing 'User' in its text, regardless of surrounding text.
Example: To find any element whose class attribute contains 'active':

```
driver.findElements(By.xpath("//*[contains(@class,'active')]"));
```

Question: How do you use **starts-with()** in an XPath expression?
Answer: The `starts-with()` function matches a node whose attribute or text starts with a given string. For instance, `starts-with(@id,'prefix')` matches elements whose `id` begins with 'prefix'. This is useful when part of an attribute is stable and the rest is dynamic. Similarly, `starts-with(text(),'Software Testing Space')` matches any element whose text begins with `Software Testing Space`.
Example: To find a label that starts with 'User':

```
driver.findElement(By.xpath("//label[starts-with(text(),'User')]"));
```

Quiz:

1. Which XPath expression matches any element with an `id` attribute containing "button"?
   A. `//*[@id='button']`
   B. `//*[contains(@id,'button')]` (Correct)
   C. `//element[contains(id,'button')]`
   D. `//element[@contains='button']`
2. What does `//a[@href='home']` select?
   A. All anchor tags with href 'home' (Correct)
   B. Only the first anchor tag with href 'home'
   C. All anchor tags with text 'home'
   D. Any tag named 'a' or 'href'
3. Which function would you use to match elements whose text starts with "Log"?
   A. `substring()`
   B. `starts-with()` (Correct)
   C. `normalize-space()`
   D. `contains()`
4. How does `//div[text()='Hello']` differ from `//div[contains(text(),'Hello')]`?
   A. The first matches partial text, second matches exact text
   B. The first matches exact text, second matches partial text (Correct)
   C. There is no difference
   D. The second is invalid syntax

3 Dynamic XPath Techniques

Question: How can you use `contains()` with an attribute in XPath for dynamic elements?
Answer: Using `contains(@attribute,'substring')` lets you match elements whose attribute value contains a given substring. This is useful for dynamic IDs or classes that change partially. For example, if a button has id `btn-save-1234`, you could use `//button[contains(@id,'btn-save')]` to match it regardless of the numeric suffix. This approach is often used when only part of an attribute is stable across sessions or builds.
Example: To find any element with a class name containing 'btn-primary':
`driver.findElement(By.xpath("//*[contains(@class,'btn-primary')]"));`

Question: How is `contains(text(),'value')` used in XPath?
Answer: The `contains(text(),'value')` function checks whether the visible text of an element contains the specified substring. This allows partial text matches, which is useful when exact text may vary or include dynamic parts. For instance, if a label reads 'Welcome, Inder', the expression `//label[contains(text(),'Welcome')]` will match it even if the name changes. This makes tests resilient to minor text changes.
Example: To find a heading containing 'Dashboard':
`driver.findElement(By.xpath("//h1[contains(text(),'Dashboard')]"));`

Question: How do you combine multiple conditions in one XPath expression?
Answer: You can combine conditions using logical `and` or `or` operators inside a predicate. For example, `//input[@type='checkbox' and @checked='checked']` finds checkboxes that are checked.
Using `or` (e.g., `//button[@id='submit' or @id='save']`) matches elements satisfying either condition. These logical operators allow building complex XPath that require multiple criteria. Make sure that you wrap each condition appropriately within the same square brackets `[...]`.
Example: To find a product listed in stock and on sale:
`driver.findElement(By.xpath("//div[contains(@class,'product') and contains(text(),'In Stock')]"));`

Question: Why are these dynamic XPath techniques useful?
Answer: Dynamic XPath techniques make your locators more robust when dealing with

changing content. By using `contains()`, `starts-with()`, and, or, etc., tests can adapt to variations in attributes or text. This reduces fragility (also called “flakiness”) in tests on pages with dynamic IDs, timestamps, or changing labels. Practically, this means fewer broken locators when the UI updates slightly, while still accurately identifying the correct elements.

View Dynamic Locators in Selenium WebDriver in my SelectorsHub tutorial here.

Quiz:

1. How would you write an XPath to select a button whose `id` contains 'login'?
   A. `//button[@id='login']`
   B. `//button[contains(@id,'login')]` (Correct)
   C. `//button[contains(text(),'login')]`
   D. `//button[starts-with(@id,'login')]`
2. Which XPath expression selects elements that contain both text 'Error' and class 'warning'?
   A. `//element[contains(text(),'Error') and contains(@class,'warning')]` (Correct)
   B. `//element[contains(text(),'Error','warning')]`
   C. `//element[@class='warning'][text()='Error']`
   D. `//element[error and warning]`
3. What is the effect of using or in an XPath predicate?
   A. It matches elements that satisfy at least one condition (Correct)
   B. It matches elements that satisfy all conditions
   C. It excludes elements that match any one condition
   D. It concatenates text values of nodes
4. Given `<span id="user_123">Name</span>`, which XPath will find this span?
   A. `//span[id='user_123']`
   B. `//span[contains(@id,'user_')]` (Correct)
   C. `//span[starts-with(text(),'Name')]`
   D. `//span[text()='user_']`

4 XPath Functions

Question: How do you use the `text()` function in XPath?
Answer: The `text()` function selects the text node of an element. It is commonly used to match element content exactly or partially. For example, `//p[text()='Hello World']` finds a paragraph whose content is exactly "Hello World". As with any text match, it is case-sensitive.
Example: Select a link with text "Home":

```
driver.findElement(By.xpath("//a[text()='Home']"));
// © Inder P Singh https://www.linkedin.com/in/inderpsingh/
```

Question: What is `normalize-space()` in XPath? When is it used?
Answer: `normalize-space()` removes leading and trailing whitespace and replaces sequences of whitespace characters with a single space in the string. It's useful when element text has unpredictable spacing or newline characters. For example, if a menu item text contains extra spaces, `//li[normalize-space(text())='Settings']` will match it by trimming the text before comparison. This avoids mismatches due to formatting characters.
Example: Find an element with trimmed text:

```
driver.findElement(By.xpath("//span[normalize-
space(text())='Submit']"));
```

Question: How can you select the last element from a group using XPath?
Answer: XPath provides the `last()` function to identify the last node in a node-set. You can use it with an index in parentheses. For example, `(//ul[@id='menu']/li)[last()]` selects the last `<li>` in the menu list. This is useful in lists or tables when you want the final item without knowing how many items there are. Similarly, you can use `position()` function to work with specific indices.
Example: Select the last row in a table:

```
driver.findElement(By.xpath("(//table[@id='data']/tr)[last()]"));
```

Question: How do `position()` and `substring()` functions work in XPath?
Answer: The `position()` function returns the index of the current node in a node-set when iterating. It is often used in predicates like `//item[position()=1]` to select the

first item. The `substring()` function extracts a part of a string, given a start and (optional) length. For example, `substring(text(),1,3)` would take the first 3 characters of the text node. Combined, these functions allow precise control, such as selecting a node based on a portion of its text content or position.

Example: Select the second item in a list:

```
driver.findElement(By.xpath("(//ol/li)[position()=2]"));
```

Question: What does `concat()` do in an XPath expression?

Answer: The `concat()` function concatenates multiple strings into one. This can be used to build a value to match against. For example, `concat('Hello ', 'World')` gives "Hello World". In XPath, it is rarely needed to locate elements, but can be used in matches, e.g., `//p[text()=concat('Test','Case')]` if text is built from parts.

Example: Match combined text

```
driver.findElement(By.xpath("//p[text()=concat('Hello','
',’User')]"));
```

Quiz:

1. Which XPath selects the second `<li>` element in an unordered list?
   A. `(//ul/li)[2]` (Correct)
   B. `//ul/li[position()=2]`
   C. `//ul/li[last()-1]`
   D. `//ul/li[contains(text(),2)]`
2. What is the result of `normalize-space(' A B ')`?
   A. `"A B"` (Correct)
   B. `" A B "`
   C. `"AB"`
   D. `"A B"`
3. How would you match an element whose text is composed by two parts "Foo" and "Bar" using `concat()`?
   A. `//div[text()=concat('Foo','Bar')]` (Correct)
   B. `//div[contains(text(),'FooBar')]`
   C. `//div[text()='FooBar']`
   D. `//div[text()=concat('Foo',' ','Bar')]`
4. Which function returns the last node in a node-set?
   A. `last()` (Correct)

B. `position()`
C. `count()`
D. `substring()`

5 XPath Axes

Question: What are **child** and **parent** axes in XPath?
Answer: The **child** axis selects direct children of a node. It is implicit when you use a forward slash. For example, `//div/child::span` or simply `//div/span` selects `<span>` elements that are direct children of `<div>`. The **parent** axis selects the immediate parent of a node using `parent::`. For example, `//li/parent::ul` selects the `<ul>` that is the parent of the `<li>`. These axes help navigate up and down one level in the hierarchy.
Example: Find the parent `<ul>` of a specific list item:

```
driver.findElement(By.xpath("//li[@id='item1']/parent::ul"));
```

I have explained XPath Axes in my SelectorsHub tutorial here.

Question: What does the **ancestor** axis do?
Answer: The **ancestor** axis selects all ancestors (parent, grandparent, and so on) of the current node up to the root. For instance, `//span/ancestor::div` selects any `<div>` that is an ancestor of a `<span>`. This is useful when you need a higher-level container of an element. To select only the nearest ancestor, you might filter by position or other criteria.
Example: Find a `<table>` element that contains a cell with a specific text:

```
driver.findElement(By.xpath("//td[text()='Price']/ancestor::table"));
```

Question: How do **descendant**, **following**, and **preceding** axes work?
Answer: The **descendant** axis selects all descendant elements of a node (children, grandchildren, etc.). For example, `//div/descendant::a` selects all links inside any `<div>`. The **following** axis selects all nodes that appear after the current node in the document, excluding descendants. For instance, `//h2/following::p` selects all paragraphs that come after an `<h2>`. The **preceding** axis is similar but selects nodes before the current node. These axes are helpful to navigate elements without being constrained to direct hierarchy.
Example: Select all links that follow a specific heading:

```
driver.findElements(By.xpath("//h2[@id='section1']/following::a"));
```

Question: What are **following-sibling** and **preceding-sibling** axes?
Answer: The **following-sibling** axis selects siblings after the current node at the same level, while **preceding-sibling** selects siblings before it. For instance, `//tr[@id='row1']/following-sibling::tr` selects all table rows after the one with id

'row1'. Conversely, `preceding-sibling::tr` selects rows before it. These are useful for moving horizontally within a table or list.

Example: Select the table row that comes immediately after a given row:

```
driver.findElement(By.xpath("//tr[@id='row1']/following-sibling::tr[1]"));
```

Follow Inder P Singh on LinkedIn to get the new AI-based Testing, Test Automation and QA documents.

Quiz:

1. Which axis would you use to select all the parent elements of a given node?
   A. `ancestor` (Correct)
   B. `child`
   C. `descendant`
   D. `following`
2. What does `//div/descendant::a` select?
   A. All `<a>` elements that are direct children of `<div>`
   B. All `<a>` elements inside `<div>`, at any depth (Correct)
   C. The first `<a>` after a `<div>`
   D. All `<a>` elements with class 'div'
3. Given `<ul><li>First</li><li>Second</li></ul>`, what does `//li[2]/preceding-sibling::li` select?
   A. "Second"
   B. "First" (Correct)
   C. Both "First" and "Second"
   D. None
4. How can you select a `<span>` that is an ancestor of `<em>` using axes?
   A. `//span/child::em`
   B. `//em/ancestor::span` (Correct)
   C. `//span/descendant::em`
   D. `//em/parent::span`

6 Chained & Complex XPath

Question: How can you build **nested predicates** in XPath?
Answer: Nested predicates allow applying multiple levels of filters. You can chain them by placing one predicate inside another or use multiple predicates in sequence. For example, `//div[@class='content'][contains(text(),'Example')]` finds a `<div>` with class 'content' and containing 'Example'. Alternatively, `//div[@id='main']//span[@class='title']` first finds a `<div>` with id 'main' and then any descendant `<span>` with class 'title'. This chaining refines selection in steps.
Example: To find a product element inside a section with a certain class:

```
driver.findElement(By.xpath("//section[@class='feature']//div[@class='product']"));
```

Question: How can you combine multiple conditions in one predicate?
Answer: Multiple conditions can be combined using `and` or `or` within the same predicate. For example, `//table/tr[td[@class='id'] and td[.='Active']]` finds table rows where one cell has class 'id' and another cell has text 'Active'. You can also stack predicates: `//*[@class='item'][contains(text(),'Active')]`. The first predicate filters by class, the second further filters the result by text. Both these techniques implement complex filters.
Example: Select a table row that has both 'completed' and 'urgent':

```
driver.findElement(By.xpath("//tr[contains(text(),'completed') and contains(text(),'urgent')]"));
```

Question: What is the strategy for chaining XPath expressions?
Answer: A common strategy is to locate a stable ancestor first, then navigate down to the target. This often uses `//` or direct child axes. For instance, identify a form by its id and then find a nested input: `//form[@id='loginForm']//input[@name='password']`. This makes locators shorter and more robust by anchoring to a known element and then narrowing down. It also avoids the long absolute paths.
Example: Find a field in a specific form:

```
driver.findElement(By.xpath("//form[@id='searchForm']//input[@type='text']"));
```

Quiz:

1. Which XPath uses two predicates to find a `<div>` with class 'card' containing 'Sale'?
   A. `//div[@class='card' and text()='Sale']` (Correct)
   B. `//div[@class='card'][text()='Sale']`
   C. `//div[@class='card'][contains(text(),'Sale')]`
   D. Both A and C
2. How does `//ul[@id='menu']//li[@class='active']` work?
   A. Finds all `<li>` with class 'active' inside any `<ul id='menu'>` (Correct)
   B. Finds the `<ul>` with id 'menu' if it has class 'active'
   C. Finds `<li>` at the root level
   D. Finds all `<li>` elements regardless of class
3. Given nested structure `<div><span><a>Click</a></span></div>`, which XPath selects the `<a>`?
   A. `//div/span/a` (Correct)
   B. `//div//a[text()='Click']`
   C. `//a[.='Click']`
   D. All of the above
4. What advantage does chaining to a stable ancestor provide?
   A. Longer XPath expressions
   B. More likely to break with UI changes
   C. Shorter and more maintainable locators (Correct)
   D. Slower test execution

Note: To expand your professional network, you're welcome to connect with Inder P Singh at https://www.linkedin.com/in/inderpsingh/

7 Best Practices in XPath Locators

Question: How can you write **meaningful** and maintainable XPath locators?
Answer: Meaningful XPaths use clear, meaningful attributes and avoid unnecessary complexity. Prefer using `id`, `name`, or `data-` attributes when available, since they are less likely to change. Avoid very long or absolute paths. For example, instead of `/html/body/div[3]/div/div[2]/table[1]`, use `//table[@id='data']`. Also, structure the XPath to reflect logical element relationships, such as `//form[@id='loginForm']//button[@type='submit']`. This makes the locator understandable at a glance.
Example: A meaningful XPath to find a 'Username' cell in a data table:

```
driver.findElement(By.xpath("//table[@id='data']//td[text()='Username']"));
```

Question: Why should you avoid **brittle** locators?
Answer: Brittle locators rely on fragile aspects of the page, such as exact element positions or auto-generated IDs. For instance, `//*[@id='main']/div[5]/span[2]` will break if any element is added or removed. Using such locators causes test failures when the UI changes slightly. Instead, find stable anchors, like labels or unique attributes. If a locator often breaks, the tests become a maintenance burden. Thus, avoiding brittle XPath reduces tests' flakiness and ongoing locators' update.

Question: What does it mean to keep XPath **maintainable**?
Answer: Maintainable XPaths are those that remain valid and understandable as the application evolves. Use consistent attribute names (like `id` or `data-testid`), shorter paths, and avoid dynamic values (like auto-generated classes or session-specific text). If multiple tests use similar elements, consider creating a shared locator strategy (such as Page Object Model) or variables. The goal is for future testers to understand and update locators easily.
Example: Good use of attribute and axis:

```
driver.findElement(By.xpath("//div[@class='menu']//a[text()='Home']"));
```

Quiz:

1. Which of the following is a **brittle** (fragile or unmaintable) XPath example?
   A. `//input[@id='email']`
   B. `//*[@id='root']/div[3]/table/tbody/tr[1]/td[4]` (Correct)
   C. `//button[text()='Login']`
   D. `//form[@id='searchForm']//input[@type='text']`
2. Why avoid using deep absolute paths?
   A. They are faster to locate elements
   B. They break easily when the page layout changes (Correct)
   C. They are required for all XPath queries
   D. They improve maintainability
3. What is a recommended practice for writing XPath?
   A. Use as many `contains()` as possible
   B. Always use absolute XPath starting at the root
   C. Rely on unique attributes and shorter relative paths (Correct)
   D. Only use text() comparisons
4. How could you make this XPath more maintainable?
   `//table[@class='table'][3]/tr[2]/td[5]`
   A. Add more indexes
   B. Use an ancestor ID or text label instead (Correct)
   C. Use `contains(text())` for every step
   D. Change it to CSS selector

8 XPath in Selenium

Question: How do you use XPath locators in Selenium WebDriver?
Answer: In Selenium (for Java, C#, etc.), XPath is used by passing it to `By.xpath()`. For example, `driver.findElement(By.xpath("//button[@id='submit']"))` finds the element matching the XPath. You can use `findElement` to get one element or `findElements` to get a list. Always confirm that your XPath string is correctly quoted in code. Using XPath in this way allows Selenium tests to interact with elements when no better selector is available.
Example: Locate a submit button by its XPath:

```
driver.findElement(By.xpath("//button[@type='submit']"));
```

Question: What are the advantages of using XPath in Selenium?
Answer: XPath is very powerful and flexible. It can navigate complex DOM relationships (like going to parent/child or using functions like `contains`). You can select elements based on any attribute, text, or position. This is particularly useful when testing dynamic web pages or when other selectors (like ID or CSS) are not available or insufficient. XPath can also select elements not directly accessible by CSS, such as elements based on their text content or selecting ancestors.
Note: View Shadow DOM explanation in my Shadow DOM SelectorsHub tutorial here.

Question: What are the disadvantages of XPath in Selenium?
Answer: XPath expressions can be slower than CSS selectors, especially in browsers where XPath is not natively optimized. Some older browsers (like older Internet Explorer) have limited XPath support. XPath syntax can be complex and verbose, making maintenance harder. Also, very complex XPaths are more prone to break if the page structure changes. As a rule of thumb, if a simple CSS selector is available, it should be preferred for speed and simplicity, but XPath is useful in other cases.

Quiz:

1. Which Selenium method uses XPath to find elements?
   A. `By.id()`
   B. `By.cssSelector()`

C. `By.xpath()` (Correct)

D. `By.tagName()`

2. Why might CSS selectors be preferred over XPath?
   A. CSS selectors cannot navigate up the DOM
   B. CSS selectors are generally faster (Correct)
   C. XPath cannot select by class
   D. XPath is not supported in Selenium
3. What is a scenario where XPath is more useful than CSS selectors?
   A. Selecting elements by ID
   B. Selecting elements by CSS class
   C. Selecting elements based on text content (Correct)
   D. Selecting elements by tag name
4. True or False: XPath can select the parent element of a given node.
   A. True (Correct)
   B. False

9 Handling Dynamic Pages

Question: How can you find elements with changing attributes like dynamic IDs?
Answer: For elements whose attributes (like id or class) change dynamically, avoid using the full dynamic attribute. Instead, use partial matches or relative context. Use `contains()` on a stable part of the attribute or find a stable parent/sibling element to anchor to. For example, if a text input has a changing ID but a consistent placeholder, one could use `//input[contains(@id,'user') or @placeholder='Enter Username']`. Another technique is to navigate via a nearby static label or container.
Example: Select an input by its label:

```
driver.findElement(By.xpath("//label[text()='Username']/following-sibling::input"));
```

Additionally, view Dynamic Locators Selenium tutorial here.

Question: What are **relative anchors** in XPath strategies?
Answer: A relative anchor means starting from a stable reference element rather than the root. For example, locate a fixed header or form and then navigate to the target element. This often uses axes or nested steps. For instance, if a form has an ID, use `//form[@id='searchForm']//input[@name='query']`. Anchors make locators resilient to layout changes outside the reference. The key is choosing elements that rarely change, such as specific labels, classes, or ids.

Question: How can you make XPath more stable on dynamic pages?
Answer: To improve stability on dynamic pages, avoid hard-coded indices and absolute paths. Use functions like `contains()`, use positional functions only when necessary, and prefer meaningful attributes. Waiting for elements (implicit/explicit waits) also ensures elements are present. If elements are injected later, you might locate their container first. Always test locators against variations of the page to ensure they hold.

Quiz:

1. Which XPath is more stable if IDs are dynamic?
   A. `//*[@id='user123']`
   B. `//input[contains(@id,'user')]` (Correct)

C. `//input[1]`
D. `//div/*`

2. If a label `<label>Email</label>` is followed by an `<input>`, how to select the input by label?
   A. `//input[preceding-sibling::label='Email']`
   B. `//label[text()='Email']/following-sibling::input` (Correct)
   C. `//label[contains(text(),'Email')]//input`
   D. `//div[@label='Email']/input`
3. Why should you avoid using index positions (like [2]) in XPath on dynamic pages?
   A. Index always starts at 0
   B. Indexed elements may shift if new elements are added (Correct)
   C. Indexed XPath is faster
   D. Because `position()` is not supported
4. What is a relative anchor?
   A. An absolute XPath from the root
   B. A base URL for navigation
   C. A stable reference element to start XPath from (Correct)
   D. A function that resets XPath context

10 Tools & Online XPath Practice

Question: What tools can help practice and build XPath expressions?
Answer: Modern browsers have built-in DevTools for inspecting elements. In Chrome or Firefox, you can right-click an element and choose "Inspect", then copy its XPath or test expressions in the console using `$x("//xpath")`. There are also browser plugins like **SelectorsHub** (view the first 5 videos in the SelectorsHub playlist) that highlight matching elements as you type. Online practice pages or custom demo apps are useful. The Software Testing Space blog provides tools like **Agile Test Estimator** and **Mock Data Generator** where testers can try out XPath queries on real web forms.

Question: How do you use DevTools to test an XPath?
Answer: In Chrome DevTools (Elements panel), you can press `Ctrl+F` and enter an XPath to see if it matches any elements on the page. Alternatively, in the Console panel, use JavaScript: e.g. `$x("//button[@id='submit']")` will return matching elements. This lets you practice, iterate and refine expressions. It's a quick way to validate your XPath without running the full test.

Question: Can you give examples of practicing XPath on provided demo tools?
Answer: Yes. For example, on the Agile Test Estimator page, there is an input for 'User Story Points to Hours Factor'. An XPath like `//input[@id='storyPointsToHoursFactor']` could locate it in DevTools. On the Mock Data Generator page, `//tr` to find the preview-table rows. By opening these pages and using DevTools or a plugin, you can iteratively build and validate XPaths against live examples.

Quiz:

1. How can you test an XPath expression in the browser without Selenium?
   A. Use the browser's Find (Ctrl+F) in Web Developer Tools (Correct)
   B. Write a Selenium script every time
   C. Install a separate application
   D. Refresh the page until it works

2. Which Chrome DevTools console command returns elements matching an XPath?
   A. `document.querySelectorAll()`
   B. `$x()` (Correct)
   C. `findElementsByXPath()`
   D. `getElementByXPath()`
3. What is the benefit of using a plugin like SelectorsHub?
   A. It automatically runs tests
   B. It highlights matched elements when you enter an XPath (Correct)
   C. It blocks XPath from running.
4. True or False: On any webpage, `Ctrl+F` in Developer Tools only searches text, not XPath.
   A. True
   B. False (Correct)

11 XPath Troubleshooting

Question: What are some common issues that you faced when using XPath locators?
Answer: Common XPath issues include syntax mistakes (missing quotes, mismatched parentheses/brackets), using incorrect quotes inside the string, or referencing non-existent attributes. Another is performance problems if using very complex or inefficient XPaths. For example, forgetting the `@` symbol in an attribute filter (`//div[class='menu']` instead of `//div[@class='menu']`) will fail. You need to make sure that each `[` has a matching `]` and strings are enclosed properly.

Question: How can you validate and debug an XPath expression?
Answer: Use the DevTools search (Ctrl+F in Elements) or SelectorsHub plugin to test the XPath directly. Tools like `$x("//xpath")` in Chrome will show matches or errors. If Selenium throws `NoSuchElementException`, try simplifying the XPath or breaking it into parts. Also watch out for iframes; an XPath won't find elements outside the current frame. You can log the result of `findElements` and check the count or use WebDriver's explicit waits to see if the element appears.

Question: What should you check if an XPath returns no results?
Answer: If an XPath returns nothing, check if the element exists at that time (maybe it loads dynamically). Verify your context (correct frame or DOM). Check for typos or dynamic values. Try validating step by step: first locate a parent element, then a child. Confirm that the quotes are correct. If using `text()`, remember it's sensitive to whitespace. Use `normalize-space()` if necessary.

Quiz:

1. If an XPath uses `@id` without quotes, what happens?
   A. It selects all elements with any id
   B. A syntax error (Correct)
   C. It works only in Firefox
   D. It ignores the id filter
2. How can you check an XPath against a page manually?
   A. Reload the page with a special parameter
   B. Use `Ctrl+F` in DevTools or `console $x()` (Correct)
   C. Try running Selenium script repeatedly
   D. Use only CSS selectors

3. What could cause an XPath to work sometimes and fail other times?
   A. Random bugs in the browser
   B. The element changes dynamically (Correct)
   C. XPath is case-insensitive
   D. The presence of CSS on the page
4. How does `normalize-space()` help in troubleshooting?
   A. It normalizes the XPath string
   B. It removes extra whitespace in text comparisons (Correct)
   C. It speeds up XPath execution
   D. It escapes special characters

12 XPath Interview Questions & Quiz

Question: What is the difference between / and // in XPath?
Answer: The single slash / means moving to a direct child in the path from the current node or root. Using // at the beginning means searching from the root down anywhere in the document, and within the path it means "select all descendants". For example, `/html/body/div` refers to an exact path, while `//div` finds all `<div>` elements in the document regardless of depth.
Example: `/html/body/div` vs `//div`

Question: How do you write an XPath to find a button with text "Submit"?
Answer: You can use the `text()` function: `//button[text()='Submit']`. This matches a `<button>` whose inner text is exactly "Submit". If case or whitespace are concerns, you could use `contains(text(),'Submit')` or `normalize-space()`.

Question: In an HTML with multiple similar elements, how do you select the third one?
Answer: Use indexing or `position()`. For example, `(//div[@class='item'])[3]` selects the third `<div>` with class 'item'. Note the parentheses are required when applying `[3]` to the result of `//div[@class='item']`. Alternatively, `(//ul[@id='list']/li)[3]` gets the third list item.
Example: `(//li)[3]`

Question: Why wouldt you use `normalize-space()` in your XPath?
Answer: `normalize-space()` XPath function is used when text content may have extra spaces or newlines. It trims and collapses whitespace. For example, if an element has text " Hello ", `//span[normalize-space(text())='Hello']` matches it even with spaces.
Example: `//span[normalize-space(text())='Hello']`

Quiz:

1. What does `//*/text()` select?
   A. All text nodes in the document (Correct)
   B. All elements with text content
   C. Only the first text node
   D. Only elements with a child text node
2. How would you select the last `<li>` in any `<ul>`?
   A. `//ul/li[last()]` (Correct)
   B. `//ul/li[position()=last()]`
   C. `(//ul/li)[last()]`
   D. All of the above
3. Which function checks if text starts with a given string?
   A. `starts-with()` (Correct)
   B. `begin()`
   C. `contains()`
   D. `prefix()`

If you find this document useful, please Like and Share this post. If you need to learn from video tutorials, you can view the complete playlists of tutorials in the YouTube channel, Software and Testing Training. Thank you!