XPath Selectors Quick Guide

Unlock the power of web scraping and automation with this essential XPath selectors cheatsheet. Covering fundamental syntax, axes, predicates, functions, and common techniques for precise element location in XML and HTML documents.

Basic XPath Selectors

Selecting Nodes

`nodename`	Selects all nodes with the specified name. **Example:** `book` selects all `<book>` elements.
`/`	Selects the root node. (Absolute path)
`//`	Selects nodes in the document from the current node that match the selection no matter where they are. (Relative path) **Example:** `//book` selects all `<book>` elements anywhere in the document.
`.`	Selects the current node. **Example:** `./p` selects `<p>` elements that are direct children of the current node.
`..`	Selects the parent of the current node.
`@attribute`	Selects the specified attribute. **Example:** `//a/@href` selects the `href` attribute of all `<a>` elements.
`*`	Selects any element node. **Example:** `/*` selects all root children.
`@*`	Selects any attribute of the current node.
`node()`	Selects any node of any kind (element, attribute, text, comment, processing instruction).

Predicates (Conditions)

`[condition]`	Filters a node-set based on the condition(s) inside the brackets.
`[1]`	Selects the first node in the node-set. **Example:** `//book[1]` selects the first `<book>` element.
`[last()]`	Selects the last node in the node-set. **Example:** `//item[last()]` selects the last `<item>` element.
`[position() < 3]`	Selects nodes with position less than 3 (i.e., the first two nodes).
`[@attribute='value']`	Selects nodes where the specified attribute equals a specific value. **Example:** `//a[@class='button']` selects `<a>` elements with `class="button"`.
`[attribute]`	Selects nodes that have the specified attribute, regardless of its value. **Example:** `//img[@alt]` selects all `<img>` elements with an `alt` attribute.
`[.= 'text']` or `[text() = 'text']`	Selects nodes whose *entire* text content equals 'text'. **Example:** `//h1[. = 'Welcome']` selects an `<h1>` with the exact text 'Welcome'.
`[condition1 and condition2]`	Combines conditions using `and`. **Example:** `//input[@type='text' and @name='username']`
`[condition1 or condition2]`	Combines conditions using `or`. **Example:** `//div[@class='error' or @class='alert']`

Combining Paths

`path1/path2`	Selects nodes matching `path2` that are direct children of nodes matching `path1`.
`path1//path2`	Selects nodes matching `path2` that are descendants (children, grandchildren, etc.) of nodes matching `path1`.
`path1 | path2`	Selects nodes matching either `path1` or `path2` (Union operator).
`(/path1/path2 | /path3/path4)[1]`	Selects the first node from the combined set of nodes matching `path1/path2` or `path3/path4`.
`//div/p | //span/a`	Selects all `<p>` elements that are direct children of a `<div>`, PLUS all `<a>` elements that are direct children of a `<span>`.
`//ul/li[1] | //ol/li[last()]`	Selects the first `<li>` in any `<ul>`, PLUS the last `<li>` in any `<ol>`.

XPath Axes & Functions

XPath Axes

`axisname::nodetest`	General syntax for selecting nodes relative to the current node.
`self::node()`	Selects the current node itself.

`child::node()`	Selects children of the current node. (Default axis, same as `./node()`). **Example:** `div/p` is shorthand for `div/child::p`
`parent::node()`	Selects the parent of the current node. (Same as `..`). **Example:** `//a/parent::div` selects the parent `<div>` of any `<a>`.
`descendant::node()`	Selects descendants (children, grandchildren, etc.) of the current node. (Same as `//node()`). **Example:** `div//p` is shorthand for `div/descendant::p`
`ancestor::node()`	Selects all ancestors (parent, grandparent, etc.) of the current node.
`following-sibling::node()`	Selects all siblings after the current node, at the same level.
`preceding-sibling::node()`	Selects all siblings before the current node, at the same level.
`following::node()`	Selects all nodes that appear *after* the current node in the document, excluding descendants.
`preceding::node()`	Selects all nodes that appear *before* the current node in the document, excluding ancestors.

Common XPath Functions

`text()`	Selects the text node(s) of the current node. **Example:** `//p/text()` selects the text content of all `<p>` elements.
`contains(string, substring)`	Checks if the first string contains the second string. **Example:** `//div[contains(@class, 'info')]` selects `<div>` elements whose `class` attribute contains 'info'.
`starts-with(string, substring)`	Checks if the first string starts with the second string. **Example:** `//input[starts-with(@id, 'user')]` selects inputs whose `id` starts with 'user'.
`ends-with(string, substring)`	(XPath 2.0) Checks if the first string ends with the second string. **Example:** `//a[ends-with(@href, '.pdf')]` selects links ending in '.pdf'.
`string(object)`	Converts the object to its string value.
`normalize-space(string)`	Removes leading/trailing whitespace and replaces internal sequences of whitespace with a single space. **Example:** `//div[normalize-space(text()) = 'Hello World']`
`count(node-set)`	Returns the number of nodes in a node-set. **Example:** `count(//li)` counts all `<li>` elements.
`position()`	Returns the position of the current node within the node-set being processed. **Example:** `//li[position() = 2]` selects the second `<li>`.
`last()`	Returns the index of the last node in the node-set being processed. **Example:** `//p[last()]` selects the last `<p>`.

Operators

`=`, `!=`, `<`, `<=`, `>`, `>=`	Comparison operators. **Example:** `//item[@price > 10]` selects items with price > 10.
`+`, `-`, `*`, `div`, `mod`	Arithmetic operators. **Example:** `//item[position() mod 2 = 0]` selects even-positioned items.
`and`, `or`	Logical operators (used in predicates). **Example:** `//a[@href and @title]` selects links with both `href` and `title`.
`|`	Union operator (combines node sets). **Example:** `//h1 | //h2` selects all `<h1>` and `<h2>` elements.

Advanced & Practical XPath

Selecting by Text Content

`//element[text()='Exact Text']`	Selects `element` nodes whose direct text child is *exactly* 'Exact Text'.
`//element[contains(text(), 'Partial Text')]`	Selects `element` nodes where the direct text child contains 'Partial Text'.
`//element[.='Exact Text']`	Selects `element` nodes whose *total* text content (including descendants) is *exactly* 'Exact Text'.
`//element[contains(., 'Partial Text')]`	Selects `element` nodes whose *total* text content (including descendants) contains 'Partial Text'. Often more useful for finding elements with specific text anywhere inside.
`//element[normalize-space(.)='Normalized Text']`	Selects `element` nodes whose total text content, after trimming whitespace, is 'Normalized Text'. Good for handling variable spacing.
`//a[text()='Click Here']`	Selects `<a>` links whose direct text is 'Click Here'.
`//button[contains(., 'Submit')]`	Selects `<button>` elements that contain the word 'Submit' anywhere in their text content.

Tips for Robust XPath

Avoid relying solely on position: `//div[2]/table[3]/tbody/tr[4]/td[1]` is fragile. The structure can easily change.
Prefer using attributes: If an element has a unique ID (`@id`), class (`@class`), name (`@name`), or other stable attribute, use it: `//input[@id='username']` , `//button[@class='submit']` . This is much more reliable.
Use text content where appropriate: For static text elements like labels, use `//label[text()='Username:']` or `//button[contains(., 'Login')]` .
Combine attributes and text: `//a[contains(@class, 'btn') and contains(text(), 'Download')]` .
Utilize axes for relationships: If an element lacks unique identifiers but a related element (like a label) does, use axes: `//label[text()='Username:']/following-sibling::input` .
Test thoroughly: Always test your XPath in a browser's developer console or dedicated tool to ensure it selects exactly what you intend and nothing else.
Handle dynamic content: For elements loaded dynamically, you might need to wait for them to appear before attempting to locate them with XPath.

CSS Selectors vs. XPath (Quick Reference)

CSS Selector	Equivalent XPath
`element`	`//element`
`#id`	`//*[@id='id']`
`.class`	`//*[@class='class']` OR `//*[contains(@class, 'class')]` (more robust for multiple classes)
`[attribute=value]`	`//*[@attribute='value']`
`[attribute~=value]` (contains word)	`//*[contains(concat(' ', @attribute, ' '), concat(' ', 'value', ' ')) ]`
`[attribute|=value]` (starts with value- or value)	`//*[starts-with(@attribute, 'value') or @attribute='value']`
`[attribute^=value]` (starts with string)	`//*[starts-with(@attribute, 'value')]`
`[attribute$=value]` (ends with string)	`//*[ends-with(@attribute, 'value')]` (XPath 2.0+)
`[attribute*=value]` (contains string)	`//*[contains(@attribute, 'value')]`
`parent > child`	`//parent/child`
`ancestor descendant`	`//ancestor//descendant`
`element:first-child`	`//element[1]`
`element:last-child`	`//element[last()]`
`element:nth-child(n)`	`//element[n]`
`element:nth-last-child(n)`	`//element[position()=last()-(n-1)]`