XPath cheat sheet

What Is XPath?

A popular functionality in web development is automation: the hands-free manipulation of a website's Document Object Model (DOM). If your target websites don't support application programming interface (API) calls directly or via Hootsuite, Buffer, or Zapier, how do you write programs to locate web elements in your browser and act upon them?

Here is where XPath plays a role. XPath, short for "XML Path Language," is a compact way of teaching a web automation engine such as Selenium WebDriver to locate elements in an HTML source code document.

XPath is compact because it represents nodes in an XML or HTML document **as a directory path** with forward slashes (/) as the main delimiter. Parking an XPath as a string rather than a standard selector path takes up less memory. Here's the same HTML element represented both ways:

Representation	HTML element in question	Character count
Selector path on an element on a different XPath cheat sheet	`body > div.body-area > main > div:nth-child(6) > div > div:nth-child(1) > div > pre > code > span:nth-child(3)`	110
The corresponding XPath	`/html/body/div[1]/main/div[6]/div/div[1]/div/pre/code/span[3]`	61

To demonstrate its compactness, the XPath string is only 55% the length of that of the selector path.

Prerequisites

As amazing as XPath appears to be, learning XPath requires a working knowledge of HTML, CSS, and JavaScript/jQuery, and the ability to open the Inspector panel in your preferred browser:

- Chrome Inspector (also applies to Chrome-based browsers such as Brave)
- Firefox Inspector
- Safari Web Inspector
- Microsoft Edge Inspector

If you're confident in the above, following the examples in this XPath cheat sheet is easier. If not, bookmark this page and come back when you're ready.

Expressions/Queries

XPath expressions include XPath queries, which typically refer to absolute and relative XPaths, and XPath function calls, which can be independent of an XML or HTML document. Make sure to distinguish XPath queries from XQuery, a query-based functional programming language outside this cheat sheet's scope yet supports XPath.

This XPath cheat sheet differs from what we're used to writing because we've found the best way to learn XPath is by looking at multiple examples and intuitively deriving the XPath pattern from them. When in doubt, use this website to test out XPath queries.

The table below presents static XPath examples, all extracted via the Inspector (see Prerequisites) from functional websites at the time of writing. The general XPath syntax follows later below.

HTML element	Selector path	Relative XPath	Absolute XPath
`<html>` tag of an HTML document	`html`	`/html`	`/html`
`<body>` tag on a website	`body`	`/html/body`	`/html/body`
Title of a website	`head > title`	`/html/head/title`	`/html/head/title`
The modifiable titular text box on a to-do list website	`#title`	`//*[@id="title"]`	`/html/body/div[1]/input`
A blue “Verify you are human” button	`#challenge-stage > div > input`	`//*[@id="challenge-stage"]/div/input`	`/html/body/div[1]/div/div[1]/div/input`
“Share Feedback” button (DuckDuckGo)	`#web_content_wrapper > div.serp__bottom-right.js-serp-bottom-`	`//*[@id="web_content_wrapper"]/div[2]/div/div/a`	`/html/body/div[2]/div[5]/div[2]/div/div/a`

	`right > div > div > a`		
Dropdown button on a website menu	`#navbarDropdown4`	`//*[@id="navbarDropdown4"]`	`/html/body/div[1]/div/nav/div/div[2]/div[1]/ul/li[4]/a`
Hyperlink portion of a Google search result	`#rso > div:nth-child(1) > div > div > div.Z26q7c.UK95Uc.jGGQ5e > div > a > h3`	`//*[@id="rso"]/div[1]/div/div/div[1]/div/a/h3`	`/html/body/div[7]/div/div[11]/div/div[2]/div[2]/div/div/div[1]/div/div/div[1]/div/a/h3`

Syntax

We have a few observations from the table above:

- The absolute XPath examples above begin with `/html`, the root (most basic, primitive parent) node of every HTML document.
- All relative XPath expressions above begin with `//*`.
    - ***Why not // as most other XPath resources say?***
    - The reason for `*` is that it's a wildcard or placeholder for the node (HTML tag, in this case) in question, as you will see shortly. You may replace `*` with a suitable HTML tag, and the XPath will still work.
- The format for getting a node with a particular ID is `//*[@id="name-of-id"]`.
- The **selector constraint** `[ ]` distinguishes between different nodes sharing the same HTML tag by their indices, such as `<div>`. For example, `div[2]` refers to the second `div` sharing the same parent node.

Hence the basic XPath syntax is as follows, reusing the to-do list example above:

XPath type	Basic XPath syntax	Example
Absolute	`/root_node/node1/node2/…/nodeN`	`/html/body/div[1]/input`
Relative	`//node1/node2/…/nodeN`	`//body/div[1]/input`
Relative, node attribute carrying a value	`//nodeX[@attribute="value"]`	`//input[@id="title"]`

What Is An XPath Axis?

The symbol `@` in XPath expressions has to do with XPath axes. An XPath axis describes a relationship to the current node on the XML/HTML hierarchy tree. The two-colon syntax (`::`) specifies conditions on the axis.

A step is an XPath segment between consecutive forward slashes (`/`), such as `html` in absolute paths. An axis can be a step.

In the table below, we leave a cell empty if no corresponding abbreviation or equivalence relationship exists. Note the symbols for `self`/`parent` axes are similar to those of the current/parent directory in scripting languages.

Axis	Abbreviation	… is short for …	Description
`ancestor`			Select all ancestors (parent, grandparent, etc.) of the current node
`ancestor-or-self`			Select all ancestors (parent, grandparent, etc.) of the current node and the current node itself
`attribute`	`@`	`@href == attribute::href`	Select all attributes of the current node
`child`		`div == child::div`	Select all children of the current node
`descendant`			Select all descendants (children, grandchildren, etc.) of the current node
`descendant-or-self`	`//`	`// == /descendant-or-self::node()/`	Select all descendants (children, grandchildren, etc.) of the current node and the current node itself
`following`			Select everything in the document after the closing tag of the current node
`following-sibling`			Select siblings (nodes with the same parent node) below the current node
`namespace`			Select all namespace nodes of the current node
`parent`	`..`	`.. == parent::node()`	Select the parent of the current node
`preceding`			Select all nodes that appear before the current node in the document, except ancestors, attribute

			nodes, and namespace nodes
`preceding-sibling`			Select siblings (nodes with the same parent node) above the current node
`self`	`.`	`. == self::node()`	Select the current node

This short table explains XPath wildcard symbols:

XPath wildcard	Description	Example
`*`	Match element node	`//a/*`
`@*`	Match attribute node; same as `attribute::*`	`//input[@*]`
`node()`	Match node of any kind	`//head/node()`
`text()`	Match text node, namely the content between `<tag>` and `</tag>`	`//title/text()`
`comment()`	Match comment node `<!-- … -->`	`//footer//comment()`
`processing-instruction()`	Match any node of the format `<?name value?>`, e.g., `<?xml catalog>`	`//*/processing-instruction()`

Selectors

XPath selectors are where XPath expressions and CSS selectors intersect. The table below illustrates the relationship between XPath axes and their corresponding CSS selectors:

XPath	CSS selector
`//div/following-sibling::p`	`div ~ p`
`//h1/preceding-sibling::[@id="wrong"]`	`#wrong ~ h1`
`//li/ancestor::ol`	`ol > li`
`//li/ancestor::ol[1]`	`ol + li`
`//ul[li]`	`ul > li`

Order selectors enclose ordinal numbers or `last()` with the selector constraint `[ ]`:

XPath with order selectors	CSS selector
`//ul/li[1]`	`ul > li:first-of-type`
`//ul/li[2]`	`ul > li:nth-of-type(2)`
`//ul/li[last()]`	`ul > li:last-of-type`
`//p[1][@id="stuck"]`	`p#stuck:first-of-type`
`//*[1][name()="a"]`	`a:first-child`
`//*[last()][name()="a"]`	`a:last-child`

Attribute selectors focus on HTML tag attributes:

XPath with attribute selectors	CSS selector
`//video`	`video`
`//button[@id="submit"]`	`button#submit`
`//*[@class="coding"]`	`.coding`
`//input[@disabled]`	`input:disabled`
`//button[@id="ok"][@type="submit"]`	`button#ok[for="submit"]`
`//section[.//h1[@id='intro']]`	`section > h1#intro`
`//a[@target="_blank"]`	`a[target="_blank"]`
`//a[starts-with(@href, '/')]`	`a[href^='/']`
`//a[ends-with(@href, '.pdf')]`	`a[href$='pdf']`
`//a[contains(@href, '://')]`	`a[href*='://']`
`//ol/li[position()>1]`	`ol > li:not(:first-of-type)`

Pro tip: You can chain XPath selectors with consecutive selector constraints, but the order matters. For example, these two XPath queries have different meanings, as explained below:

- `//a[1][@href='/']`
    - Get the first `<a>` tag and check its `href` has the value `'/'`.
- `//a[@href='/'][1]`
    - Get the first `<a>` with the given `href`.

Predicates

You can use logical operators in XPath queries:

Operator	Description	Example
`|`	Union: join two XPath expressions	`//a | //span`
`+`	Addition	`2 + 3`
`-`	Subtraction	`3 - 2`
`*`	Multiplication	`2 * 5`
`div`	Division	`5 div 2`
`=`	Equal	`number(//p/text())=9.80`
`!=`	Not equal	`number(//p/text())!=9.80`
`<`	Less than	`number(//p/text())<9.80`
`<=`	Less than or equal to	`number(//p/text())<=9.80`
`>`	Greater than	`number(//p/text())>9.80`
`>=`	Greater than or equal to	`number(//p/text())>=9.80`
`or`	or	`//div[(x and y) or not(z)]`
`and`	and	`//div[@id="head" and position()=2]`
`mod`	Modulus (division remainder)	`5 mod 2`

Functions

The table below illustrates functions used in XPath expressions. Some, such as `boolean()`, are standalone XPath expressions. Some of the following appear in the examples above.

Function	Description	Example
`name()`	Return the name of the node (e.g., HTML tag)	`//*/a/../name()`
`text()`	Return the inner text of the node, excluding the text in child nodes	`//div[text()="Submit?"]/*/text()`
`lang(str)`	Determine whether the context node matches the given language (Boolean)	`//p[lang('en-US')]`
`namespace-uri()`	Return a string representing the namespace URI of the first node in a given NodeSet. This function applies to XML documents.	`//*[@*[namespace-uri()='http://foo.example.com']]`
`count()`	Count the number of nodes in a NodeSet and return an integer	`//table[count(tr)=1]`
`position()`	Return a number equal to the context position from the expression evaluation context	`//ol/li[position()=2]`
`string()`	Convert an argument to a string	`string(//div)`
`number()`	Convert an object to a number and return the number	`number(//img/@width)`
`boolean()`	Evaluate an expression and return true or false. Use this to check for the existence of nodes/attributes.	`boolean(//div/a[@class="button"]/@href)`
`not(expression)`	Evaluates Boolean NOT on an `expression`	`button[not(starts-with(text(),"Submit"))]`
`contains(first, second)`	Determine whether the `first` string contains the `second` string (Boolean)	`//button[contains(text(),"Go")]`
`starts-with(first, second)`	Check whether the first string begins with the second string (Boolean)	`//[starts-with(name(), 'h')]`
`ends-with(first, second)`	(Only supported in XPath 2.0; Selenium supports up to XPath 1.0) Check whether the first string ends with the second string (Boolean)	`//img[ends-with(@src, '.png')]`
`concat(x,y)`	Concatenate two or more strings `x,y` and return the resulting string.	`//div[contains(concat(' ',normalize-space(@class),' '),' foobar ')]`

	The example checks if the attribute `foobar` is part of a space-separated list.	
`substring(given_string, start, length)`	Return a part of a `given_string` beginning from the `start` value with a specified `length`	`substring("button", 1, 3)`
`substring-before(given_string,substring)`	Return a string that is part of a `given_string` before a given `substring`	`substring-before("01/02", "/")`
`substring-after(str,substring)`	Return a string that is part of a `given_string` after a given `substring`	`substring-after("01/02", "/")`
`translate()`	Evaluate a string and a set of characters to translate and return the translated string	`translate('The quick brown fox.', 'abcdefghijklmnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')`
`normalize-space()`	Remove redundant white space characters and return the resulting string	`normalize-space(' hello  world  !   ')`
`string-length()`	Return a number equal to the number of characters in a given string	`string-length('hello world')`

Pro tip: You can use nodes inside functions. Examples:

- `//ul[count(li) > 2]`
  - Check if the number of `<li>` tags inside the `<ul>` tag is greater than two.
- `//ul[count(li[@class='hide']) > 0]`
  - Check the number of `<li>` tags with class “`hide`” inside the `<ul>` tag is a positive integer.

More Usage Examples

Here’s how to extract data from a specific element:

XPath	Description
`//span/text()`	Get the inner text of the `<span>` tag. In the example below, `"Click here"` is the result. `<span>Click here</span>`
`//*/a[@id="attention"]/../name()`	Find the name of the parent element to an `<a>` tag with `id="attention"`
`//body//comment()`	Get the first comment under the `<body>` tag.

Extracting data from multiple elements is straightforward. The following XPath expressions apply to the same HTML example:

```
<div>
  <a class="pink red" href="http://banks.io">oranges</a>
  <a class="blue" href="http://crime.io">and lemons</a>
  <a class="green" href="http://skyscraper.io">apple</a>
  <a class="violet" href="http://leaks.io">honey</a>
  <a class="amber" href="http://technology.io">mint</a>
  <input type="submit" id="confirm">Go!</input>
</div>
```

XPath	Description

`//a/@href`	Get the URLs (the `href` string value) in all `<a>` tags:
	
	`http://banks.io`
	`http://crime.io`
	`http://skyscraper.io`
	`http://leaks.io`
	`http://technology.io`

`//a/text()`	Get the inner text of all `<a>` tags:
	
	`oranges`
	`and lemons`
	`apple`
	`honey`
	`mint`

`//a/@class`	Get the classes of all `<a>` tags:
	
	`pink red`
	`blue`
	`green`
	`violet`
	`amber`

The table below shows ways to extract data from an element based on its attribute value—note the mandatory use of `@` in the final step of each XPath query:

XPath query	Description
`//a[@href="http://skyscraper.io"]/@class`	Get the class in the `<a>` tag where the `href` string value is `"http://skyscraper.io"`: `green`
`//*[contains(@class, "red")]/@href`	Get the URL (the `href` string value) in any tag with the class `'red'`: `http://banks.io/`
`//input[@id="confirm"]/@type`	Get the `type` attribute of an `<input>` tag with `id="confirm"`: `submit`

If you want to extract data from an element based on its position, check out these examples:

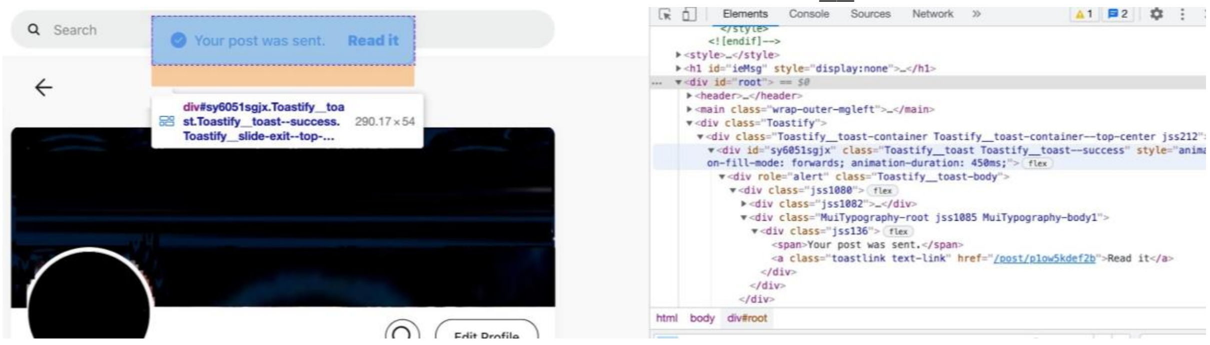
XPath query	Description
`//table/tbody/tr[3]`	Get the third `<tr>` element in a table
`//a[last()]`	Get the last `<a>` tag in the document
`//main/article/section[position()>2]/h3`	Get the `<h3>` tags in all `<section>` tags after the second instance of `<section>`

Now that you've made it to the last section of this cheat sheet, here are three real-life XPath examples of XPath in Selenium.

1. Absolute XPath expressions to get the "Accept All Cookies" footer bar out of the way:
   - `cookiespress="/html/body/div[1]/main/div/div/div/div/div[4]/div/div/div[2]/div/button[1]"`
   - `loginwith="/html/body/div[1]/main/div/div/div/div/div[1]/div[1]/div[2]/form/div[1]/span"`

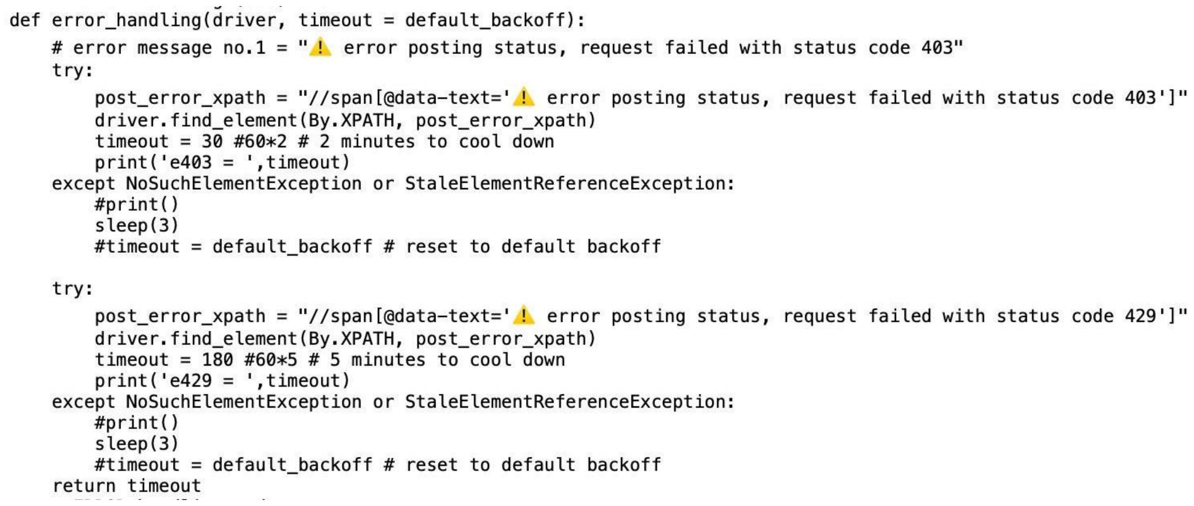
```
cookiespress="/html/body/div[1]/main/div/div/div/div/div[4]/div/div/div[2]/div/button[1]"
loginwith="/html/body/div[1]/main/div/div/div/div/div[1]/div[1]/div[2]/form/div[1]/span"

while True:
    try:
        Accept_All_Cookies = driver.find_element(By.XPATH, cookiespress)
        break
    except NoSuchElementException:
        sleep(0.1)
ActionChains(driver).move_to_element(Accept_All_Cookies).click(Accept_All_Cookies).perform()
```

2. This relative XPath expression maps to a pop-up triggered when a user successfully posts to a certain social media platform: `"//*[@class='Toastify__toast--success']"`



```
    </style>
    <![endif]-->
  ▸<style>…</style>
  ▸<h1 id="ieMsg" style="display:none">…</h1>
… ▾<div id="root"> == $0
    ▸<header>…</header>
    ▸<main class="wrap-outer-mgleft">…</main>
    ▾<div class="Toastify">
      ▾<div class="Toastify__toast-container Toastify__toast-container--top-center jss212">
        ▾<div id="sy6051sgjx" class="Toastify__toast Toastify__toast--success" style="anima
        on-fill-mode: forwards; animation-duration: 450ms;"> flex
          ▾<div role="alert" class="Toastify__toast-body">
            ▾<div class="jss1080"> flex
              ▸<div class="jss1082">…</div>
              ▾<div class="MuiTypography-root jss1085 MuiTypography-body1">
                ▾<div class="jss136"> flex
                    <span>Your post was sent.</span>
                    <a class="toastlink text-link" href="/post/plow5kdef2b">Read it</a>
                  </div>
                </div>
              </div>
```

3. The following Python function handles XPath error messages `"//span[data-text='⚠️ error posting status, request failed with status code 403/429']"`:



```
def error_handling(driver, timeout = default_backoff):
    # error message no.1 = "⚠️ error posting status, request failed with status code 403"
    try:
        post_error_xpath = "//span[@data-text='⚠️ error posting status, request failed with status code 403']"
        driver.find_element(By.XPATH, post_error_xpath)
        timeout = 30 #60*2 # 2 minutes to cool down
        print('e403 = ',timeout)
    except NoSuchElementException or StaleElementReferenceException:
        #print()
        sleep(3)
        #timeout = default_backoff # reset to default backoff

    try:
        post_error_xpath = "//span[@data-text='⚠️ error posting status, request failed with status code 429']"
        driver.find_element(By.XPATH, post_error_xpath)
        timeout = 180 #60*5 # 5 minutes to cool down
        print('e429 = ',timeout)
    except NoSuchElementException or StaleElementReferenceException:
        #print()
        sleep(3)
        #timeout = default_backoff # reset to default backoff
    return timeout
```

Conclusion

We hope this comprehensive XPath cheat sheet helps you accelerate your IT learning journey, especially in application development and security. You can read about XPath injection attacks and testing for it. For more information, check out our blog articles on coding and our resources on development, security, and operations (DevSecOps) below:

https://courses.stationx.net/p/the-complete-application-security-course
https://courses.stationx.net/p/cyber-security-python-and-web-applications
https://courses.stationx.net/p/web-hacking-become-a-web-pentester