XPath

This is an XPath selectors cheat sheet, which lists commonly used XPath positioning methods and CSS selectors

XPath Selectors

Getting started

- Xpath test bed (whitebeam.org)

Test in Firefox or Chrome console:

```
$x('/html/body')
$x('//h1')
$x('//h1')[0].innerText
$x('//a[text()="XPath"]')[0].click()
```

Descendant selectors

Xpath	CSS
`//h1`	h1
`//div//p`	div p
`//ul/li`	ul > li
`//ul/li/a`	ul > li > a
`//div/*`	div > *
`/`	:root
`/html/body`	:root > body

Order selectors

Xpath	CSS
`//ul/li[1]`	ul > li:first-child
`//ul/li[2]`	ul > li:nth-child(2)
`//ul/li[last()]`	ul > li:last-child
`//li[@id="id"][1]`	li#id:first-child
`//a[1]`	a:first-child

Attribute selectors

Xpath	CSS
`//*[@id="id"]`	#id
`//*[@class="class"]`	.class
`//input[@type="submit"]`	input[type="submit"]
`//a[@id="abc"][@for="xyz"]`	a#abc[for="xyz"]
`//a[@rel]`	a[rel]
`//a[starts-with(@href, '/')]`	a[href^='/']
`//a[ends-with(@href, '.pdf')]`	a[href$='pdf']
`//a[contains(@href, '://')]`	a[href*='://']
`//a[contains(@rel, 'help')]`	a[rel~='help']

Siblings

Xpath	CSS
`//h1/following-sibling::ul`	h1 ~ ul
`//h1/following-sibling::ul[1]`	h1 + ul
`//h1/following-sibling::[@id="id"]`	h1 ~ #id

jQuery

Xpath	CSS
`//ul/li/..`	$('ul > li').parent()
`//li/ancestor-or-self::section`	$('li').closest('section')
`//a/@href`	$('a').attr('href')

Misc selectors

Xpath	CSS
`//h1[not(@id)]`	h1:not([id])
`//button[text()="Submit"]`	Text match
`//button[contains(text(),"Go")]`	Text contains (substring)
`//product[@price > 2.50]`	Arithmetic
`//ul[*]`	Has children
`//ul[li]`	Has children (specific)
`//a[@name or @href]`	Or logic
`//a | //div`	Union (joins results)

XPath Expressions

Steps and axes

`//`	`ul`	`/`	`a[@id='link']`
Axis	Step	Axis	Step

Prefixes

Prefix	Example	Means
`//`	`//hr[@class='edge']`	Anywhere

Prefix	Example	Means
/	/html/body/div	Root

Axes

Axis	Example	Means
/	//ul/li/a	Child
//	//[@id="list"]//a	Descendant

XPath Predicates

Predicates

```
//div[true()]
//div[@class="head"]
//div[@class="head"][@id="top"]
```

Restricts a nodeset only if some condition is true. They can be chained.

Operators

```
# Comparison
//a[@id = "xyz"]
//a[@id != "xyz"]
//a[@price > 25]


# Logic (and/or)
//div[@id="head" and position()=2]
//div[(x and y) or not(z)]
```

Using nodes

```
# Use them inside functions
//ul[count(li) > 2]
//ul[count(li[@class='hide']) > 0]

# Returns `<ul>` that has a `<li>` child
```

```
//ul[li]
```

Indexing

```
//a[1]                 # first <a>
//a[last()]            # last <a>
//ol/li[2]             # second <li>
//ol/li[position()=2] # same as above
//ol/li[position()>1] #:not(:first-child)
```

Use `[]` with a number, or `last()` or `position()`.

Chaining order

```
a[1][@href='/']
a[@href='/'][1]
```

Order is significant, these two are different.

Nesting predicates

```
//section[.//h1[@id='hi']]
```

This returns `<section>` if it has an `<h1>` descendant with `id='hi'`.

XPath Functions

Node functions

```
name()            # //[starts-with(name(), 'h')]
text()            # //button[text()="Submit"]
                  # //button/text()
lang(str)
namespace-uri()


count()           # //table[count(tr)=1]
position()        # //ol/li[position()=2]
```

String functions

```
contains()        # font[contains(@class,"head")]
starts-with()     # font[starts-with(@class,"head")]
ends-with()       # font[ends-with(@class,"head")]


concat(x,y)
substring(str, start, len)
substring-before("01/02", "/")  #=> 01
substring-after("01/02", "/")   #=> 02
translate()
normalize-space()
string-length()
```

Boolean functions

```
not(expr)         # button[not(starts-with(text(),"Submit"))]
```

Type conversion

```
string()
number()
boolean()
```

XPath Axes

Using axes

```
//ul/li                       # ul > li
//ul/child::li                # ul > li (same)
//ul/following-sibling::li    # ul ~ li
//ul/descendant-or-self::li   # ul li
//ul/ancestor-or-self::li     # $('ul').closest('li')
```

`//`	`ul`	`/child::`	`li`
Axis	Step	Axis	Step

Steps of an expression are separated by `/`, usually used to pick child nodes. That's not always true: you can specify a different "axis" with `::`.

Child axis

```
# both the same
//ul/li/a
//child::ul/child::li/child::a
```

`child::` is the default axis. This makes `//a/b/c` work.

```
# both the same
# this works because `child::li` is truthy
//ul[li]
//ul[child::li]


# both the same
//ul[count(li) > 2]
//ul[count(child::li) > 2]
```

Descendant-or-self axis

```
# both the same
//div//h4
//div/descendant-or-self::h4
```

`//` is short for the `descendant-or-self::` axis.

```
# both the same
//ul//[last()]
//ul/descendant-or-self::[last()]
```

Other axes

`ancestor`		
`ancestor-or-self`		
`attribute`	`@`	`@href` is short for `attribute::href`
`child`		`div` is short for `child::div`
`descendant`		
`descendant-or-self`	`//`	`//` is short for `/descendant-or-self::node()/`
`namespace`		
`self`	`.`	`.` is short for `self::node()`
`parent`	`..`	`..` is short for `parent::node()`
`following`		
`following-sibling`		
`preceding`		
`preceding-sibling`		

There are other axes you can use.

Unions

```
//a | //span
```

Use `|` to join two expressions.

XPath More examples

Examples

```
//*                 # all elements
count(//*)          # count all elements
```

```
(//h1)[1]/text()    # text of the first h1 heading
//li[span]          # find a <li> with an <span> inside it
                    # ...expands to //li[child::span]
//ul/li/..          # use .. to select a parent
```

Find a parent

```
//section[h1[@id='section-name']]
```

Finds a `<section>` that directly contains `h1#section-name`

```
//section[//h1[@id='section-name']]
```

Finds a `<section>` that contains `h1#section-name`. (Same as above, but uses descendant-or-self instead of child)

Closest

```
./ancestor-or-self::[@class="box"]
```

Works like jQuery's `$().closest('.box')`.

Attributes

```
//item[@price > 2*@discount]
```

Finds `<item>` and check its attributes

Also see

Devhints (devhints.io)
Xpath test bed (whitebeam.org)