XPath
Cheat Sheet

About this Cheat Sheet

* All the XPath expressions I’m gonna cover on this Cheat Sheet will be applied on the HTML markup I’ve added after this page.

HTML web page

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <title>XPath and CSS Selectors</title>
    </head>
    <body>
      <h1>XPath expressions simplified</h1>
      <div class="intro">
        <p>
           I'm paragraph within a div with a class set to
intro
        <span id="location">I'm a span with ID set to
location and i'm within a paragraph</span>
        </p>
        <p id="outside">I'm a paragraph with ID set to
outside and i'm within a div with a class set to intro</p>
      </div>
       <p>Hi i'm placed immediately after a div with a class
set to intro
       </p>
       <span class='intro'>Div with a class attribute set to
intro
      </span>
      <ul id="items">
        <li data-identifier="7">Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
        <li>Item 4</li>
      </ul>

      <a href="https://www.google.com">Google</a>
      <a href="http://www.google.fr">Google France</a>
      <p class='bold italic'>Hi, I have two classes</p>
      <p class='bold'>Hi i'm bold</p>
    </body>
</html>
```

BASICS

An element is a **tag** in the HTML markup.

Example:

The **'p'** tag aka paragragh is called an element.

To select any element from HTML web pages we simply use the following syntax

Example:

To select all **p** elements we can use the following XPath selector

```
//p
```

Although this approach works perfectly fine, it's not recommended to use it, because if for example we want only to select the "**p**" elements that are inside the first div with a class attribute equals to "**intro**" this approach won't be the best solution, this is why we always prefer to target elements either by their **class** attribute, **id** or by **position** so we can limit the scope of the XPath expression.

CLASS & ID

So to select any element by its **class** attribute value we use the following syntax:

```
//elementName[@attributeName='value']
```

Example:

Let's say we want to select the "**p**" elements that inside the "**div**" with a class attribute equals to "**intro**" in this case we use the following XPath expression:

```
//div[@class='intro']/p
```

If we want to select the "**p**" element with "**id**" equals to "**outside**" we can use the following XPath expression:

```
//p[@id='outside']/p
```

REMEBER:

Please note, the same exact class attribute value can be assigned to more than one element however, and id can be assigned to only and only one element.

Sometimes we want also to select elements based on a foreign attribute which doesn't belong to HTML markup standard. For example to select the "**li**" element with the attribute "**data-identifier**" equals to 7 in this case we use the following XPath expression:

```
//li[@data-identifier="7"]
```

Sometimes the element we want to select does have two classes, for example, to select the "**p**" element with a class attribute equals to "**bold**" and "**italic**" in this case we use the following XPath expression:

```
//p[@class='bold italic']
```

OR:

Although the element does have two classes we can for example search for a **substring** within the class attribute value by using the **contains** function.

```
//p[contains(@class, 'italic')]
```

REMEBER:

The contains function takes two arguments:

- **The first one is where to search, whether on the class attribute value, id or anything else.**
- **The second argument is the value you're looking for.**
- **The value you search for is also case sensitive, so be careful!**

Value lookup

Let's say you want to select all the "**a**" elements in which the "**href**" attribute value starts with "**https**" and not "**http**", in this case we can use the following XPath expression:

```
//a[starts-with(@class, 'https')]
```

So search for the text at the beginning we use the caret sign "**starts-with**" function which takes the same arguments as the contains function.

Now if you want to search for a value at the end we use the "**ends-with**" function, however, this function is not supported on XPath version 1.0 which is the version used by the majority of the browsers and LXML.

Finally if we want to search for a particular value in between we use the contains function as explained before.

If you want to get the text of a particular element you can use the text function, for example, to get the **text element** of the "**p**" element with **id** equals to "**outside**" we use the following XPath expression:

```
//p[@id="outside"]/text()
```

The position

Okay, let's say you want to get the second "**li**" element from the "**ul**" element with "**id**" equals to "**outside**", in this case you can use the following XPath expression:

```
//ul[@id="items"]/li[2]
```

However, if you want to select the second list item but you also want to make that its text element is "Item 2", in this case you can use the following XPath expression:

```
//ul[@id="items"]/li[position() = 2 and text() = "Item 2"]
```

Notice in this case I did use the **position()** function, the **text()** function plus the "**and**" logical operator.

In contrast to the "**and**" logical operator we also have the "**or**" logical operator.

REMEBER:

In XPath everything we write within [] is known as a predicate.

XPath axes

In XPath an axis is used to search for an element based on its relatioship with another element, we have some axes which we can use to navigate **up** and **down** in the HTML markup.

All axes in XPath use the follwing syntax:

```
ElementName::axis
```

XPath axes (GOING UP)

- **The parent**
  - The parent axis is used to get the parent of a specific element, for example the get the parent of the "**p**" element with id equals to "outside" we use the following XPath expression:

```
//p[@id="outside"]/parent::node()
```

  - The **node()** function in XPath is used to get the "element" no matter what its type is.

- **The ancestor**
  - The ancestor axis can be used to all the ancestors of a specific element, for example to get the ancestors(**parent**, **grand parent**, ...) of the "**p**" element with id equals to "**outside**" we use the following XPath expression:

```
//p[@id="outside"]/ancestor::node()
```

- **The preceding**
  - In XPath the preceding axis will get all the elements that do precede an element **excluding its ancestors**.

- **Preceding sibling**
    - In XPath the preceding-sibling axis will get the sibling that do precede an element, in other words it will return the brother that is on the top of a specific element.

XPath axes (GOING DOWN)

To go down on the HTML markup we also have 4 axis which are:

- **The child** axis which will get the children of a specific element
- **The following** axis which will return all the elements that are after the **closing tag** of a specific element.
- **The following-sibling** axis which will return all the elements that are after the closing tag of an element **but** these elements should share the same parent.
- **The descendant** axis which will get the descendants of a particular element.