**CSS selector**

Class: can be shared among multiple elements

.<class name>

ID: unique to only one element

#<id name>

<tag name>.<class name>

<tag name>#<ID name>

Two strings in a class name

.<string1>.<string2>

Other tag attribute

[<attribute name> = <value>]

<tag name>[<attribute name>=<value>]

Value starts with some given string

[<attribute name> ^= <string>]

Value ends with some given string

[<attribute name> $= <string>]

Value contains some string in the middle

[<attribute name> ~= <string>]

Child element

<Parent tag name>{.<Parent class name>} <Child tag name>

Element immediately after another element

<First element identifier> + <Immediate element identifier>

Element after another element (there can be other elements between the two)

<First element identifier> ~ <Other element identifier>

All child elements within a parent element

<Parent element identifier> > <Child elements identifier>

Get the elements in order

<Element tag name>:nth-child(<index>)

Get odd elements

<Element tag name>:nth-child(odd)

Get even elements

<Element tag name>:nth-child(even)

Get first elements at the same time

<Element 1> , <Element 2> , ......

**Xpath Selector – Come with functions and logical operators for richer criteria specification**

select tag

//<tag name>

select with tag and its attribute value

//<tag name>[@<attribute name> = <attribute value>]

select with tag and its attribute value starts with/ ends with/ contains certain string using functions

//<tag name>[starts-with(@<attribute name>, <string>)]

//<tag name>[ends-with(@<attribute name>, <string>)]

//<tag name>[contains(@<attribute name>, <string>)]

select with tag and its text value starts with/ ends with/ contains certain string using functions

//<tag name>[starts-with(text(), <string>)]

//<tag name>[ends-with(text(), <string>)]

//<tag name>[contains(text(), <string>)]

select with tag and a few possible attributes using logical or operator

//<tag name>[@<attribute 1> = <value1> or @<attribute 2> = <value 2>]

select based on order positions

//<tag name>[<position index>]

select based on union of a few order positions using *position()* function and mathematical order operators

//<tag name>[position() = <index 1> or position() = <index 2>]

//<tag name>[position() >= <first index> and position() <= <last index>]

specify the last item using function *last()*

//<tag name>[position() = last()]

hierarchical selection

//<parent tag>/>child tag>

get specific parent of an element

//<tag name>/parent::<parent tag>

get parent of an element when the tag name of the parent is unknown

//<tag name>/parent::node()

get all ancestors (all earlier generations in the hierarchy) of an element

//<tag name>/ancestor::node()

get all ancestors and the element itself

//<tag name>/ancestor-or-self::node()

get all elements that come before an element but excluding all its ancestors

//<tage name>/preceding::node()

get preceding sibling elements (preceding elements in the same parent element(

//<tag name>/preceding-sibling::node()

get children elements

//<tag name>/child::<children tag>

//<tag name>/child::node()

get elements after an element, including sub-elements

//<tag name>/following::node()

get sibling elements after an element (a way to remove all sub-elemnents that appear after)

//<tag name>/following-sibling::node()

get all descendants of an element (including all children down the hierarchy)

//<tag name>/descendant::node()

\* text of, for example an href element, is considered a child of the href element. Therefore, by using descendant, you would extract both the href element as well as its text.

Combining selection criterion using logical operator and. Example:

//li[position() = 1 and contains(text(), “hello”)]

\*Select the list element at index 1, with its text containing the text “hello”