Advanced CSS Selector

Tag Selector / Type selector

It helps us to select a particular tag of elements

```
div {
 /**/
}


p {
 /**/
}
```

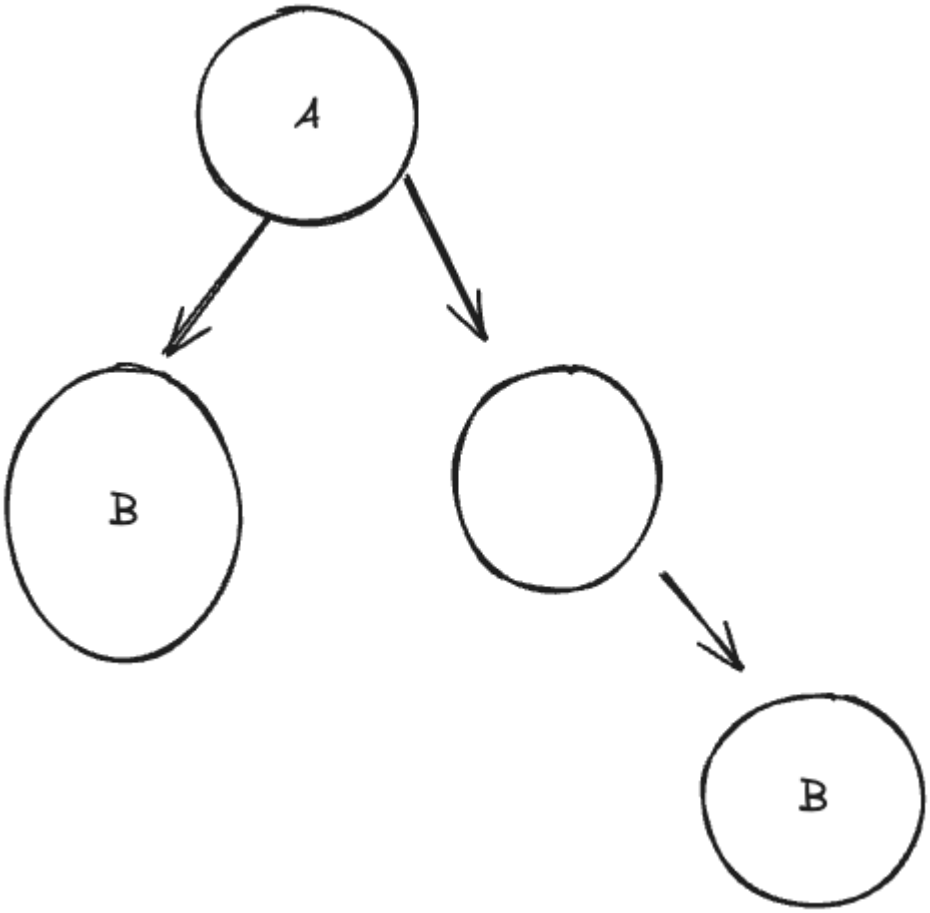
Descendant Selector

If we want to select any element inside the DOM subtree of other element we can use descendant selector.

```
A B {

}
```

`A B` refers that any B i.e. present inside the subtree of A. A and B can be id selectors or tag selectors.



ID selectors

This set of selectors can actually help us to target / select any element based on the `id` attribute. To write id selector we put a `#` behind the id value of the element : `#id_value`

```
<plate></plate>
<plate id="middle"></plate>
<plate></plate>
```

So, in the above code if we just want to select the plate with id middle, then

```
#middle {
        /* ... */
}
```

Combining ID selector and Descendants

We can combine the id selector and descendant selector. Say we have to target any element which is in the subtree of a parent with a given id.

```
<div>
        <bento></bento>
        <plate id-"fancy">
                <pickle />
        </plate>
        <plate>
                <pickle />
        </plate>

</div>
```

SO now if we want to select all the pickle tags present inside the plates with id `fancy`.

```
#fancy pickle {
        /** ... */
}
```

Class selector

It selects all the set of elements having the same class. To use a class selector we put a dot behind the classname in the css like this: `.className`.

```
<apple/>
<apple class="small"/>
<plate>
  <apple class="small"/>
</plate>
<plate/>
```

Say, we want to select all the apples with a small class.

```
.small {

}
```

Because only apple tags are there with a small class, we can just say `.small`. If any other tags also would have a small class and we were required to select only those apples which have got small class then we need to make a couple of changes.

So assume we have the HTML like this:

```
<apple/>
<apple class="small"/>
<bento>
  <orange class="small"/>
</bento>
<plate>
  <orange/>
</plate>
<plate>
  <orange class="small"/>
</plate>
```

Now here both apple and orange tag has got a small class. If we just want to select those oranges which has a small class then:

```
orange.small {

}
```

Here we are combing the class selector with any other selector, by putting them side to side without a space.

For example:

```
ul.important {

}
```

The above code means that we want to select only those ul which have got important class. If there are any other tags apart from ul which has got important class we don't want them.

So, for example:

```
<bento>
  <orange/>
</bento>
<orange class="small"/>
```

```
<bento>
  <orange class="small"/>
</bento>
<bento>
  <apple class="small"/>
</bento>
<bento>
  <orange class="small"/>
</bento>
```

Say, we want to select all those oranges which have got a small class and are present inside bento.

```
bento orange.small {

}
```

Here we have combined class selector with descendant combinator, making sure that bento is the parent and any orange with a small class inside the bento is selected.

Comma combinator

Comma combinator selects multiple elements together by putting their selectors in a comma separated fashion.
It works something like this:

```
A , B, C {

}
```

Here A, B and C are some selectors and we have put them comma separated that means the styling mentioned will be applied to all of them.

```
<pickle class="small"/>
<pickle/>
<plate>
  <pickle/>
</plate>
<bento>
  <pickle/>
</bento>
<plate>
```

```
  <pickle/>
</plate>
<pickle/>
<pickle class="small"/>
```

Say, we want to select all the plates and bentos.

```
bento, plate {
        /* selected both bento and plates */
}
```

Universal selector

This selector selects everything on the page. To use it we put a `*` and then it means we have triggered universal selector.

```
* {
}
```

The above code selects all the elements on the page.

We can combine universal selector and descendant combinator as well to make sure that we select everything present inside a parent.

```
<plate id="fancy">
  <orange class="small"/>
</plate>
<plate>
  <pickle/>
</plate>
<apple class="small"/>
<plate>
  <apple/>
</plate>
```

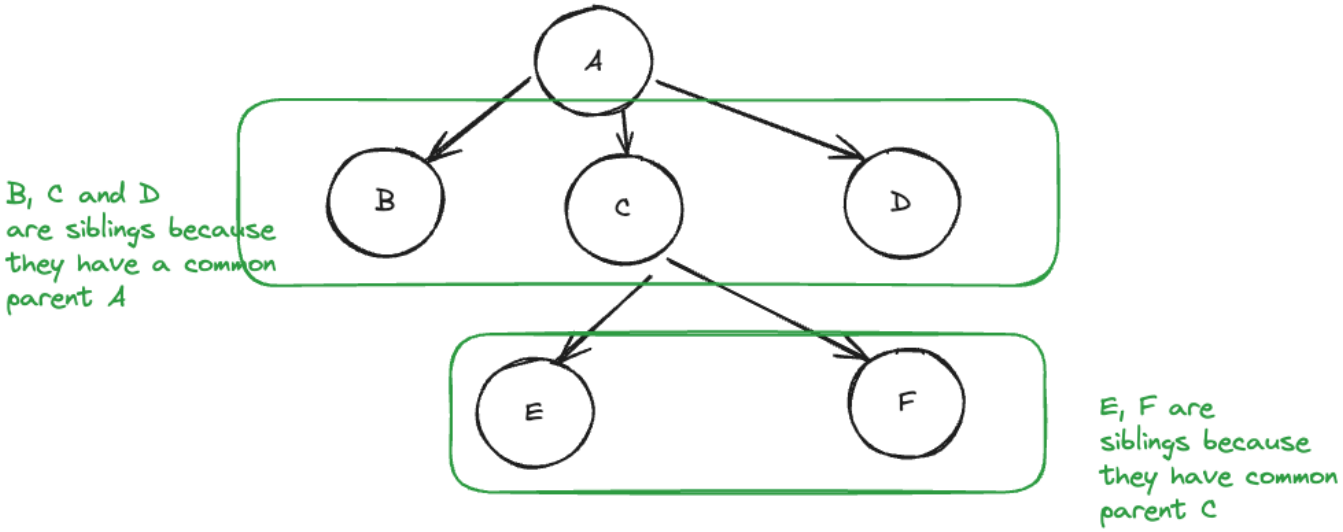
Now what if we want to select everything on a plate ?

```
plate * {
}
```

When we have `plate` space * then the space means we are using descendant combinator and we want everything inside a plate.

Adjacent sibling selector

Adjacent sibling selector is initiated using the `+` symbol.
What is a sibling ? Sibling are those elements which have got the same immediate parent.



Now if we want to select adjacent sibling, i.e. select an element that directly follows another element we can use `+` symbol.

```
C + D {

}
```

Select all `D` elements that directly follow (directly adjacent) to C. Elements that follow one another are siblings and are on the same level as well as have got same parent.

```
<bento>
  <apple class="small"/>
</bento>
<plate />
<apple class="small"/> <!-- This apple is next to a plate -->
<plate />
<apple/> <!-- This apple is next to a plate -->
<apple class="small"/> <!-- This apple is next to an apple -->
<apple class="small"/>
```

Say, we want to select every apple that is next to a plate

```
plate + apple {

}
```

General sibling selector

Now we don't have a constraint of immediate following or adjacent placement, we just want to select any sibling w.r.t another one we can use this symbol : `~`

```
B ~ D {

}
```

You can select the D sibling next to B, not necessarily adjacent.

```
<pickle/><!-- This is not selected because it doesn't follow bento -->
<bento>
  <orange class="small"/>
</bento>
<pickle class="small"/> <!-- This is selected -->
<pickle/><!-- This is selected -->
<plate>
  <pickle/>
</plate>
<plate>
  <pickle class="small"/>
</plate>
```

We want to select the pickles beside the bento

```
bento ~ pickle {

}
```

Child selector

If at any point of time we want to select immediate child of a parent we can use child selector which has an angle bracket as the symbol : `>`

```
A > B {

}
```

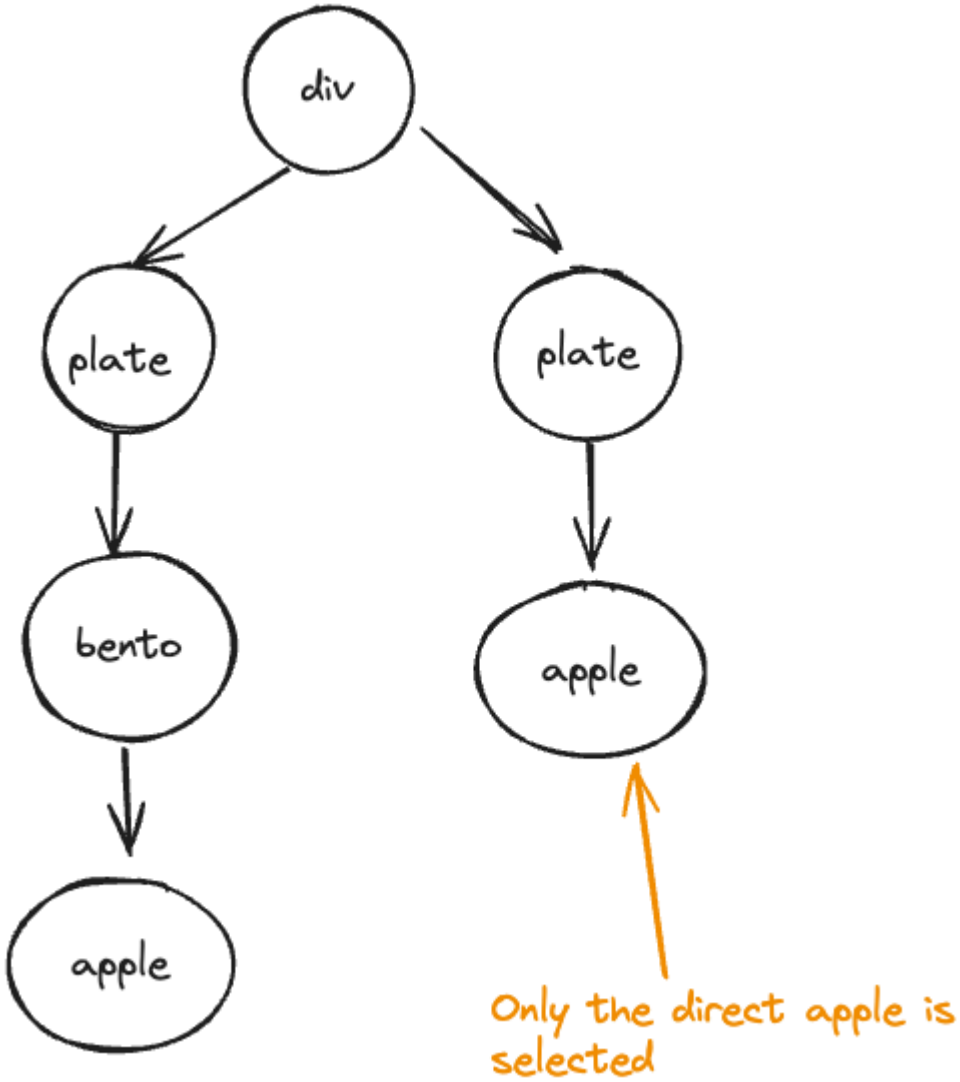
This means select the B elements which are immediate child of A element. We can also say direct children of A.

```
<plate>
  <bento>
        <apple/> <!-- Here apple is not a direct child of plate -->
  </bento>
</plate>
<plate>
  <apple/> <!-- Here apple is a direct child of plate -->
</plate>
<plate/>
<apple/>
<apple class="small"/>
```

If we want to select the apple which is present directly on the plate, then we can use child selector.

```
plate > apple {

}
```

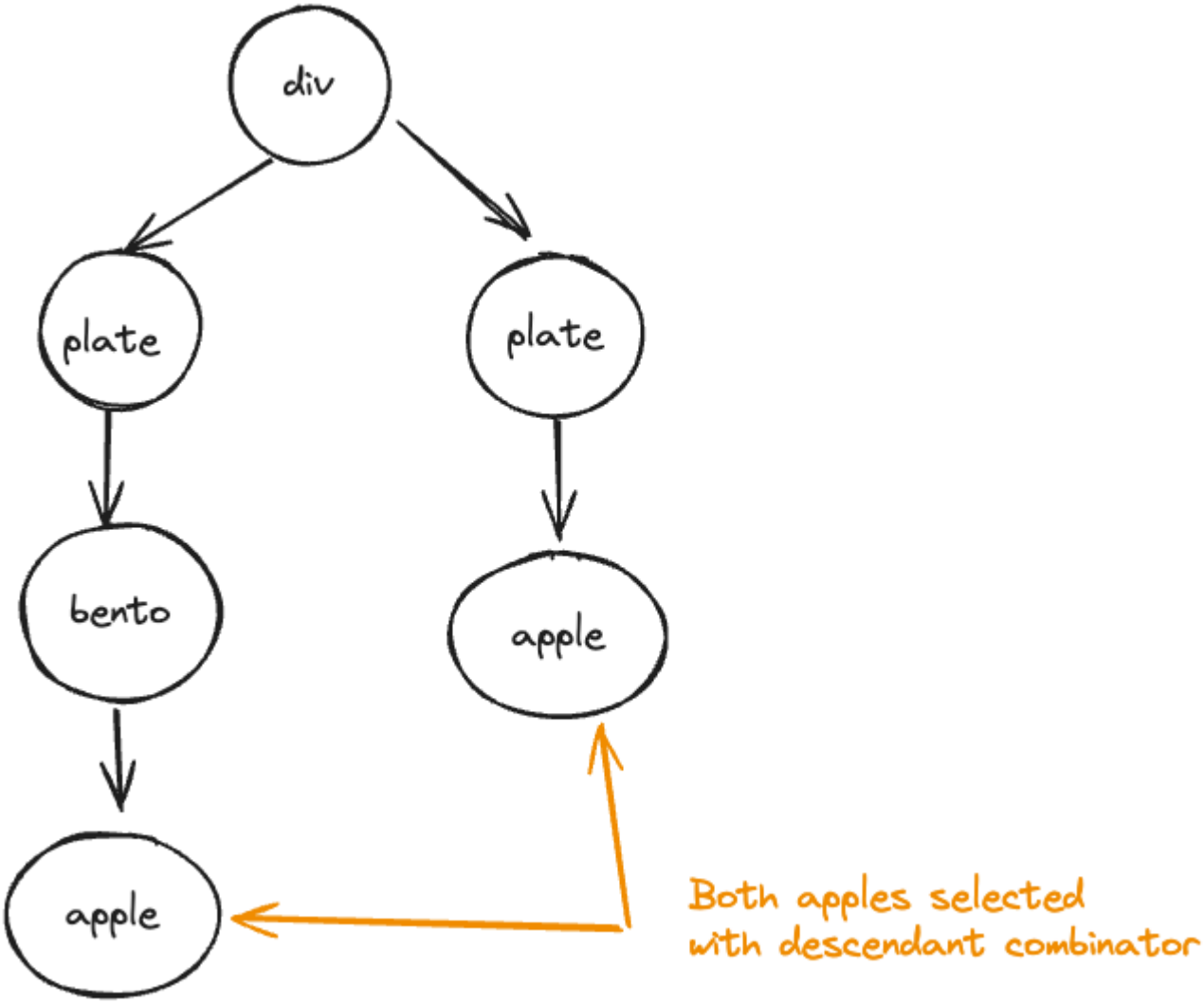


You might think, what will happen if I do

```
plate apple {

}
```

Here we are not selecting direct child, instead we are selecting all the apples which are descendant of a plate, so both the apples above will be selected which is not desired.



First child pseudo selector

It selects the first child element inside a parent / another element. To use it we say `element:first-child` . It means select all the elements which are first-child of their parent. If we say

```
p:first-child {

}
```

select all the p tags which are first-child of a parent.
Child is any element that is directly present inside a parent.

```
<bento/>
<plate />
<plate>
```

```
  <orange /><!-- This orange will be selected -->
  <orange />
  <orange />
</plate>
<pickle class="small" />
```

Say, we want to select the topmost orange on a plate.

```
orange:first-child {

}
```

It selects those oranges which are the first-child of their parent.

Last child pseudo selector

Just like the first-child, we have a `last-child` pseudo selector as well that does the same thing as first-child but only targets the last element of a parent instead of first.

```
orange:last-child {

}
```

This will select only those oranges which are the last child of a parent.

Only child pseudo selector

You can select any element that is the only element inside the parent. To use it we say `element:only-child` and it selects only those elements which are the only child present inside the parent.

```
span:only-child {

}
```

It selects those span tags that are the only element of some parent.

```
<plate>
  <apple/>
</plate>
```

```
<plate>
  <pickle />
</plate>
<bento>
  <pickle />
</bento>
<plate>
  <orange class="small"/>
  <orange/>
</plate>
<pickle class="small"/>
```

Say, we want to select the apple and the pickle on the plates
Here we cannot just say `pickle:only-child` because as an only child picke is present in the bento and plate both, So we need to be specific.

```
apple:only-child , plate > pickle{

}
```

or

```
apple:only-child , plate pickle:only-child {

}
```

Both of the above ways will select the right pickle for us.

Nth child pseudo selector

If we don't want just the first or the last element of a parent, but instead want a custom one like nth child from start, we can use nth child pseudo selector.

```
<plate/> <!-- first -->
<plate/> <!-- second -->
<plate/> <!-- third -->
<plate id="fancy"/> <!-- forth -->
```

Say we want to select the third plate.

```
plate:nth-child(3) {

}
```

Nth last child pseudo selector

Just like the nth child from start, we have an nth child from last selector. To use it we say `element:nth-last-child(n)` .

```
<plate/>
<bento/>
<plate>
  <orange/>
  <orange/>
  <orange/>
</plate>
<bento/>
```

So here if we want to select the first bento we can either do `bento:nth-child(2)` or `bento:nth-last-child(3)` .

First of type Selector

This is going to select first element of it's type among a group of sibling elements.

```
span:first-of-type {
        /* among some siblings it will select the first span present */
}
```

For example:

```
<orange class="small"/>
<apple/>
<apple class="small"/>
<apple/>
<apple class="small"/>
<plate>
  <orange class="small"/>
  <orange/>
</plate>
```

Say, we need to select the first apple among all the siblings

```
apple:first-of-type {

}
```

Just like first-of-type we have a last-of-type selector also that just selects the last element of a type among the siblings.

Nth of Type Selector

This is going to select elements based on the position among the siblings. The position value can be a number or we can say even / odd to select either all the even occurrences or odd occurrences.

```
div:nth-of-type(2) {
        /* this will select the 2nd div among the siblings */
}

div:nth-opf-type(odd) {
        /* This will select the odd occurences of the div among the
siblings*/
}
```

For example: say we want to select all the even plates:

```
<plate/> <!-- odd -->
<plate/><!-- even -->
<plate/><!-- odd -->
<plate/><!-- even -->
<plate id="fancy"/><!-- odd -->
<plate/><!-- even -->
```

We can do it using:

```
plate:nth-of-child(even) {
}
```

Here we can also put a formula as an argument which looks like `an + b`.
This means select every ath element starting from the bth element.

```
<plate/>
<plate>
  <pickle class="small" />
</plate>
<plate> <!-- this will be picked -->
  <apple class="small" />
</plate>
<plate/><!-- this will be picked -->
<plate>
  <apple />
</plate>
<plate/>
```

Say, we want to select every second plate starting from the 3rd plate

```
plate:nth-of-type(2n + 3) {

}
```

Only of Type Selector

Select elements that are the only ones of their type within of their parent element
Examples
`p span:only-of-type` selects a span within any `p` if it is the only span in there.

```
<plate id="fancy">
  <apple class="small" />
  <apple />
</plate>
<plate>
  <apple class="small" />
</plate>
<plate>
  <pickle />
</plate>
```

Say, we want to collect the apple on the middle plate, now here we have apple on both middle and first plate, but on the first plate apple is not the only unique element, hence it should be easily solvable by only-of-type

```
plate apple:only-of-type {

}
```

Empty Selector

Select elements that don't have children. Say we have `div:empty` then it will only select those divs which don't have a child.

Negation Pseudo-class

Select all elements that don't match the negation selector. We can use it to select everything except something.
Example 1:

```
:not("#fancy") {
        /* this will select all those elements which are not having fancy
id"
}
```

Example 2:

```
<plate id="fancy">
  <apple class="small" />
</plate>
<plate>
  <apple />
</plate>
<apple />
<plate>
  <orange class="small" />
</plate>
<pickle class="small" />
```

Here lets say we want to only select big apple, so here can't just say `:not(".small")`, because due to this it will select all those elements which donot have a small class, hence oranges and plates will be also collected. Instead we can do:

```
apple:not(".small") {
/* This only selects apples which don't have a small class */*
```

```
}
```

More examples:

- `div:not(:first-child)` selects every div that is not a first child.
- `:not(.big, .medium)` selects all elements that do not have class="big" or class="medium".

Attribute Selector

Select all elements that have a specific attribute. We can have a lot of variables where sometimes we only look for presence of an attribute, or may be we look for a specific value in the attribute etc.

- [xyz] to select all the elements that have got the attribute xyz in them
- [xyz="something"] to select all the elements that have got value as something to the xyz attribute
- [xyz^="abc"] Select all elements with an attribute xyz whose value starts with abc
- [xyz$="abc"] Select all elements with an attribute xyz whose value end with abc
- [xyz*="abc"] Select all elements with an attribute xyz whose value contains abc