CSS Selectors - again!

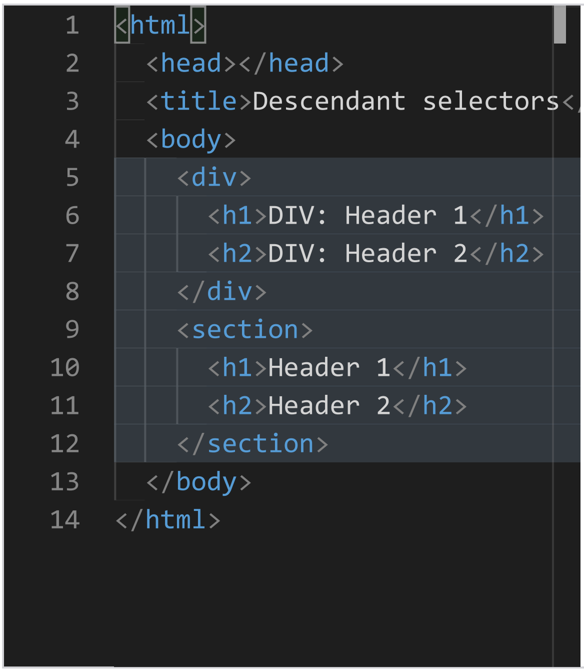
In this lesson, we will take a look at descendant selectors, combinator selectors and pseudo class selectors. While these may seem strange to you now, I'll show you the important bits you need to know.

Descendant Selectors

In the last lesson, we took a look at selectors in CSS. Here, I'll introduce another form of css selectors.

Let's see an example.

Consider the basic `html` markup below:



```
 1  <html>
 2    <head></head>
 3    <title>Descendant selectors<
 4    <body>
 5      <div>
 6        <h1>DIV: Header 1</h1>
 7        <h2>DIV: Header 2</h2>
 8      </div>
 9      <section>
10        <h1>Header 1</h1>
11        <h2>Header 2</h2>
12      </section>
13    </body>
14  </html>
```

The markup above is fairly simple. The important bit is between line 5 and 12.

You'd notice that the `div` on line 5 houses two header elements `h1` and `h2`. The same may be said of the `section` on line 9.

Generally, the `html` DOM can be represented in terms of parent-child relationship. What I mean by that is, since the `div` on line 5 houses the `h1` and `h2` elements, the `div` may be called their `parent` element. Consequently, the header elements, `h1` and `h2` may be referred to as `child` elements.

CSS avails us a way to select elements based off of their DOM relationship. A pivotal one is the `descendant` selector - which is based on parent and child relationship.

Let’s see an example.

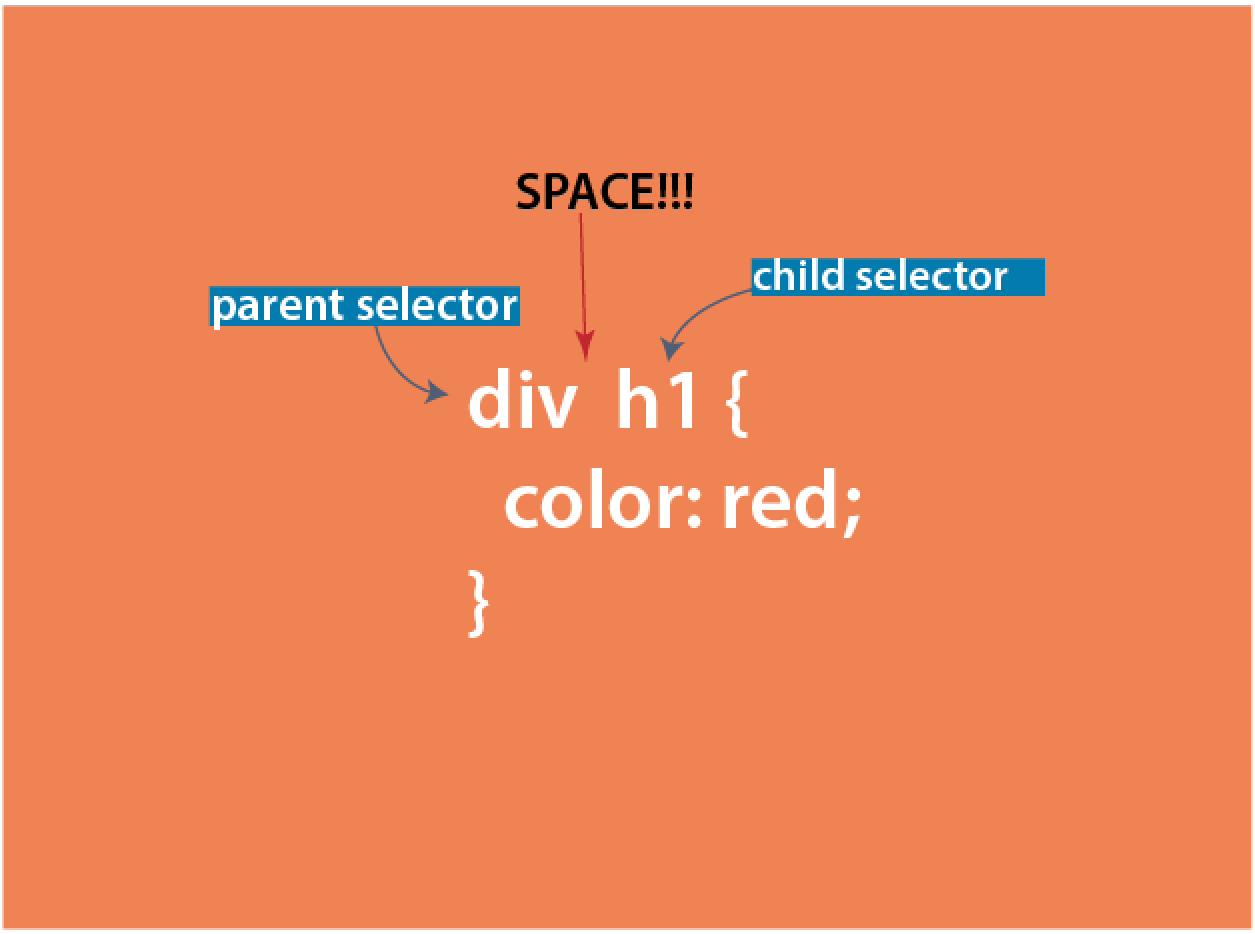
In the markup discussed above, how do you select the `h1` and `h2` within the `div` only?

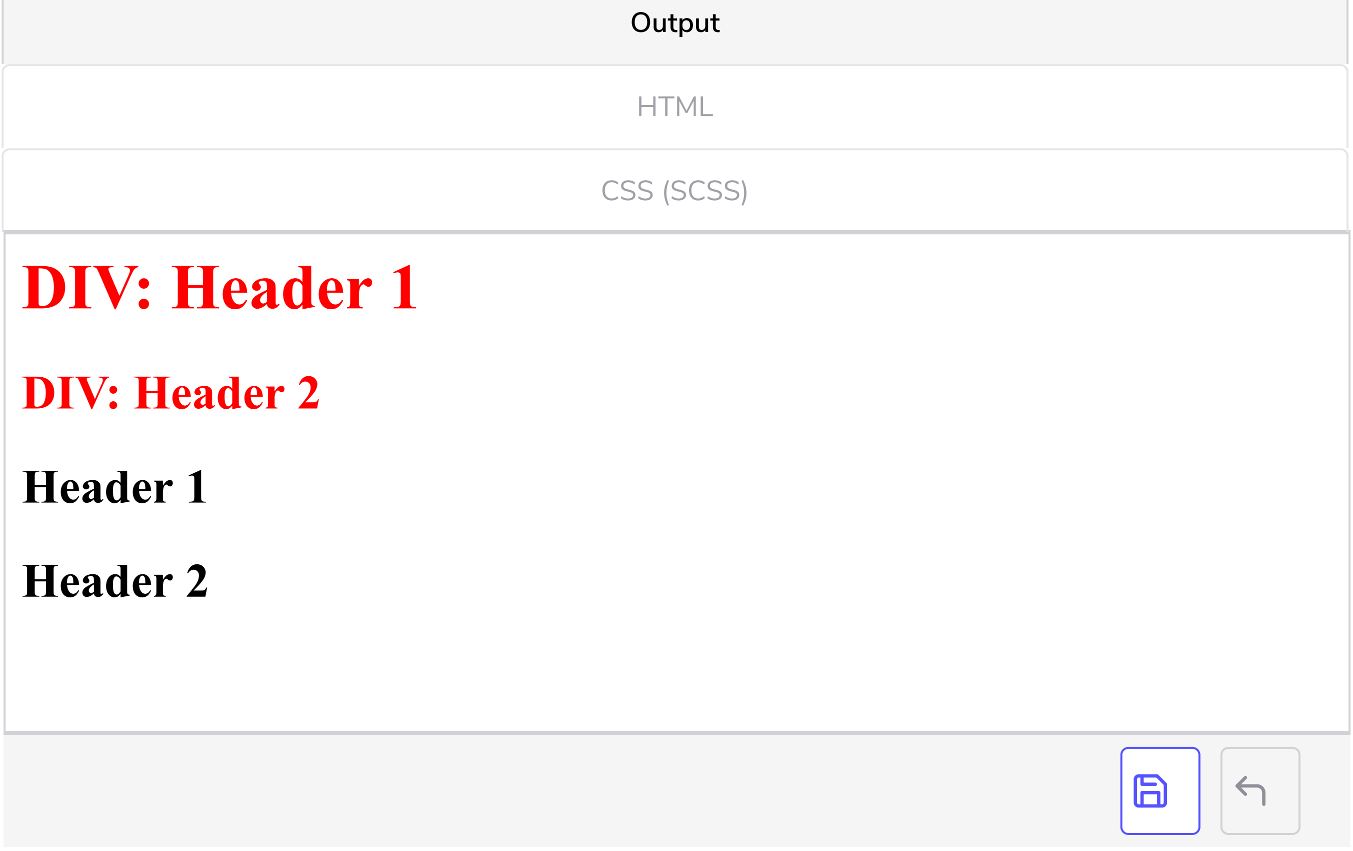
Using descendant selectors, do this:

```
div h1 {
  color: red;
}

div h2 {
  color: red;
}
```

The CSS above will ONLY select the `h1` and `h2` within the `div`. The other `h1` and `h2` within the `p` tag will be left unstyled.





DIV: Header 1

DIV: Header 2

Header 1

Header 2

As you can see above, the headers within the `section` are left unstyled. `Header 1` and `Header 2` are left unstyled! Interesting.

Now go ahead and style the header elements within the section.

Here’s my solution:

```
section h1,
section h2 {
  color: blue;
}
```

You see what I have done there?

To avoid repetition, I have written multiple statements and styled them.

That summarizes what you need to know about descendant selectors.

Lets take a look at another form of CSS selector.

Pseudo-class Selectors

Consider the basic `html` document below:

```
<html>
  <head></head>
  <title>Descendant selectors</title>
  <title>Descendant selectors</title>
  <body>
    <a href="www.google.com"> Click Me </a>

  </body>
</html>
```

and the style below:

```
a:link {
    color: #0000ff;
}
a:visited {
  color: #ff00ff;
}
a:hover {
  color: #00ccff;
}
a:active {
  color: #ff0000;
}
```

Okay, what have I done there?

Here's the bit you are familiar with:

```
a {
     color: red;
 }
```

The usual tag selector with color style defined.

So, what are Pseudo-class selectors?

The pseudo-class selectors targets the selector, **in a specific state.**

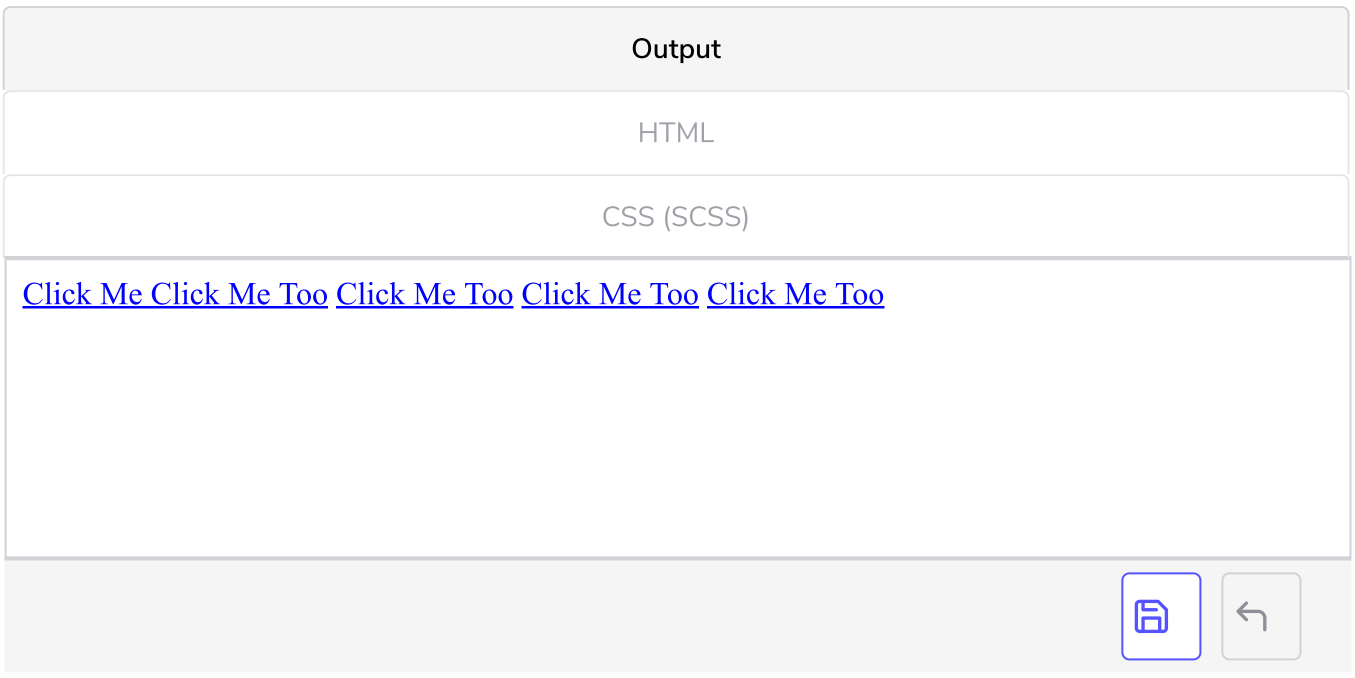
In the example above `a:link` will target and style every `a` tag with an `href` attribute. i.e an `a` that contains a link.

`a:visited` will target every anchor tag, `a` that has already been **visited** (clicked) on the page.

`a:hover` will target every link as you **hover** over them

Finally, `a:active` will style the link, just when you click on it. When it is **active**

Now play with the output below to see the code above at work. Hover over the links, click them, and see how they are styled.



```
Click Me Click Me Too Click Me Too Click Me Too Click Me Too
```

Done playing around with the output above?

Notice how the color changes when you hover over the links. Also, note how the links you have clicked change in color.

Do not forget to click the `CSS` tab in the code output above.

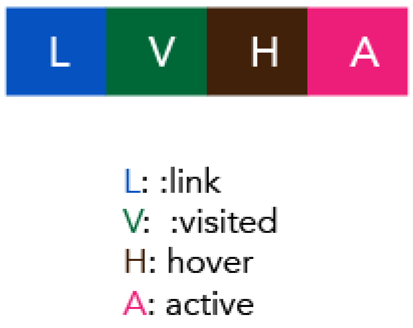
Let’s discuss the order for which these link states should be written.

The Order for Link Pseudo-Selectors

If you took a look at the CSS tab above, you’ll notice that I have written the link pseudo-selectors in a particular order.

`:link`, followed by `:visited`, `:hover` and finally, `:active`.

The popular acronym, **LVHA** may be of help. LVHA, kinda like LoVe HA!



In summary, the order in which you define these link pseudo-selectors is important. It should follow the order LVHA i.e `:link`, `:visited`, `:hover`, and finally `:active`

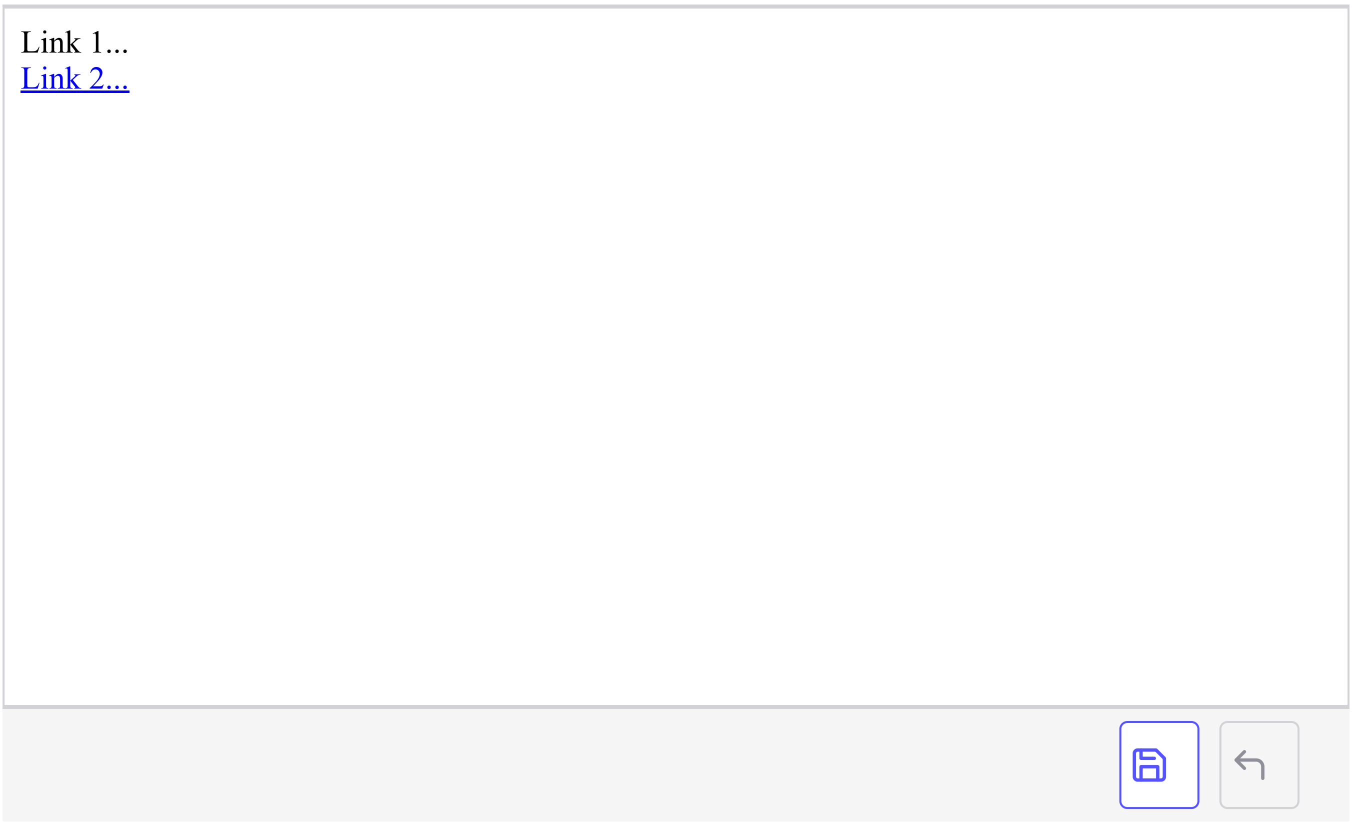
Exercise

The following is a basic html markup:

Output

HTML

CSS (SCSS)


Link 1...
Link 2...

The exercise is to style these links using the pseudo-class selectors discussed above. Note that one of the links has NO `href` attribute.

Does this make any difference when styling the anchor tags? Take a look for yourself!

Other Pseudo-classes you should be aware of.

If you skipped the exercise above, please go back and get it done. I'll wait.

There's more to pseudo-classes than just styling links. The beauty of Pseudo-classes is the ease it brings to common styling issues. Some of this issues we will tackle in the practical sections that come along.

It's going to be fun. For now, let's get you comfortable with the essentials.

First Child

In a much earlier example, we took a look at the parent child relationship in the html DOM. The selector, `first-child` says it all. It targets the first child of a specific element within the parent element.

An example is always great. Let’s see one.

Example:

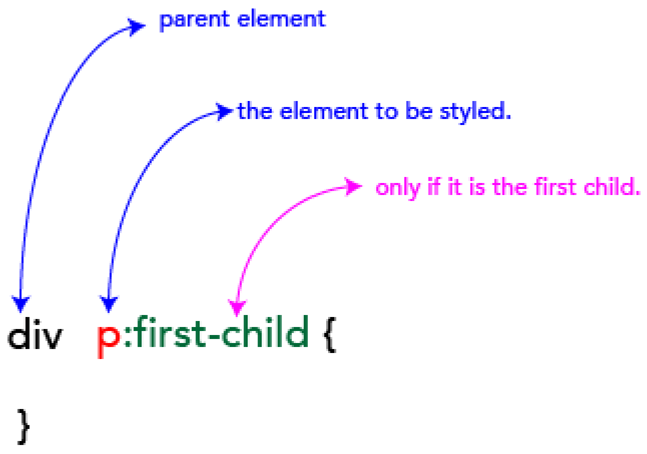
Consider the following `html` markup:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>A Simple Page</title>
<link rel="stylesheet" href="styles.css" />
</head>
<body>
    <div>
        <p>I am the first paragraph here</p>
        <p>I am the second paragraph here</p>
        <p>I am the third paragraph here</p>
        <p>I am the last paragraph here</p>
    </div>
</body>
</html>
```

We have a `div` with four `p` tags.

To style the first `p` tag using the `:first-child` pseudo selector, do this:

```
div p:first-child {
   color: red
}
```



In this example, I have added the `div` parent selector.

In this particular case, you can do this too (without the parent selector)

```
p:first-child {
   color: red;
}
```

Last Child

The `:last-child` pseudo-class selector is the opposite of `:first-child`. While `:first-child` targets the first child, `:last-child` targets the last child.

```
div p:last-child {
  color: red;
}
```

Only Child

There was `first-child`, `last-child`, now `only-child`?

You may be wondering why all this is important. If you're going to make a good soldier, it is important to know the weapons available to you.
The same goes for styling the web too.

The `:only-child` pseudo-class selector selects an element if it is the only child of it's parent.

Here is an example:

```
<!DOCTYPE html>
    <html lang="en">
    <head>
    <meta charset="utf-8" />
    <title>A Simple Page</title>
    <link rel="stylesheet" href="styles.css" />
    </head>
    <body>
       <ul>
          <li>Item one</li>
          <li>Item two</li>
          <li>Item three</li>
       </ul>
       <ul>
           <li>Lone poor child</li>
       </ul>
    </body>
    </html>
```

In the markup above, the `li` in the second `ul` is the **only child** element. It can be selected and styled accordingly, like so:

```
li:only-child {
  color: red;
}
```

Nth Child

Okay, I promise this is the last of its kind :-)

Imagine that instead of using `first-child`, `last-child` or `only-child`, you can use `nth-child` i.e "n" could be anything? 1,2, 3, 4 etc?

That's cool, and powerful, right? That, and even more does `nth-child` make available.

Example

Consider the basic markup below:

Consider the basic markup below:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>A Simple Page</title>
<link rel="stylesheet" href="styles.css" />
</head>
<body>
    <ul>
        <li>Item one</li>
        <li>Item two</li>
        <li>Item three</li>
        <li>Item four</li>
        <li>Item five</li>
        <li>Item six</li>
        <li>Item seven</li>
    </ul>
</body>
</html>
```

The first `li` may be selected like this:

```
li:nth-child(1) {
  color: red
}
```

The `nth-child` is smart to know that `1` meant the first child. You may change the numeric value to suit your cause.

One more thing…

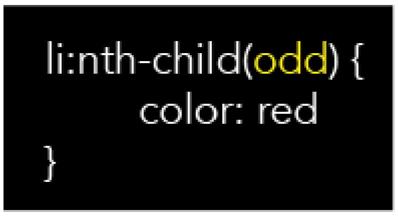
While this looks cool, the `nth-child` is even more powerful.

It allows for selecting multiple elements dynamically.

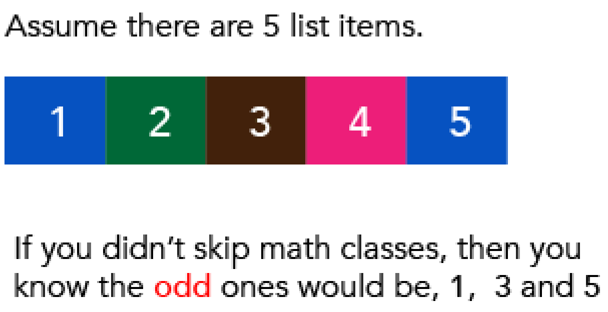
What do I mean by “multiple” elements?

I could easily target every `odd` or `even` list item, `li` like this:

```
li:nth-child(odd) {
   color: red;
}
```



```
li:nth-child(odd) {
        color: red
}
```



Assume there are 5 list items.

If you didn't skip math classes, then you know the odd ones would be, 1, 3 and 5

The `nth-child` selector is smart enough to decipher the `odd` or `even` child elements.

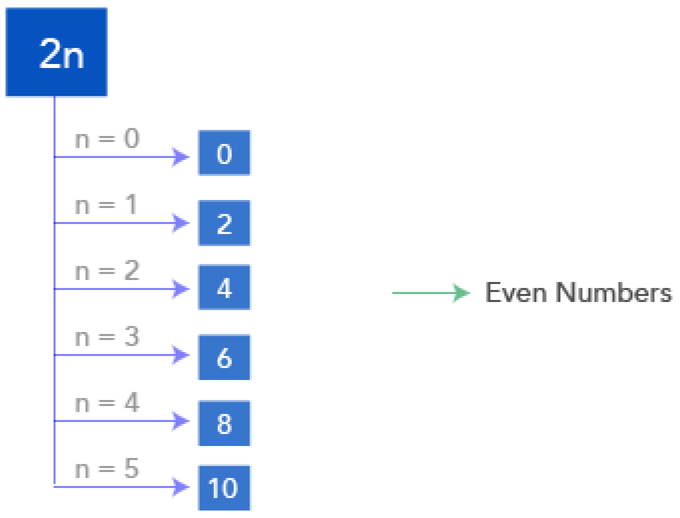
The `nth-child` selector still has one more trick up its sleeves.

Using basic math, we can also use multiplier expressions.

They look like this:

```
li:nth-child(2n) {
   color: red;
}

li:nth-child(2n+1) {
   color: red;
}
```



In the practical sections that come, we will be taking a closer look at practical use cases for the `nth-child` pseudo-class selector.

Brace up!

Exercise

(a) The `html` markup below should look familiar.

Output

HTML

CSS (SCSS)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
```

```
<title>A Simple Page</title>
<link rel="stylesheet" href="styles.css" />
</head>

<body>
    <div>
        <p>I am the first paragraph here</p>
        <p>I am the second paragraph here</p>
        <p>I am the third paragraph here</p>
        <p>I am the last paragraph here</p>
    </div>
    <div>
        <p>Lone poor paragraph</p>
    </div>
</body>
</html>
```

For this exercise, be sure to write your css in the tab above.

Todo:

1. Using `:first-child`, give the first paragaph a `color` of `red` and a `font-size` of `10px`
2. Using `:last-child`, give the last paragaph a `background-color` of `red` and a `color` of `white`
3. Using `:only-child`, give the 'lone poor' paragaph a `background-color` of `blue` and a `color` of `white`

(b) Consider the markup below:

Output

HTML

CSS (SCSS)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title>A Simple Page</title>
<link rel="stylesheet" href="styles.css" />
```

```
<link rel="stylesheet" href="styles.css" />
</head>
<body>
  <ul>
      <li>1</li>
      <li>2</li>
      <li>3</li>
      <li>4</li>
      <li>5</li>
      <li>6</li>
      <li>7</li>
      <li>8</li>
      <li>9</li>
      <li>10</li>
  </ul>
</body>
</html>
```

4. Style the `odd` and `even` list items using the multiplier expressions, `2n` and `2n+1`. Do NOT use the keywords, `odd` or `even` in your css style declarations.