

Cascading Style Sheet (CSS)

Part 03

CSS Layout - The position Property

The position property specifies the type of positioning method used for an element.

- static (default property of HTML element)
- relative
- fixed
- absolute
- sticky

position : static

- it is always positioned according to the normal flow of the page
- elements are not affected by the `top`, `bottom`, `left`, and `right` properties.

```
div.static {  
    position: static;  
    border: 3px solid green;  
    width: 250px;  
}
```

```
<div class="static">  
    This div element has position: static;  
</div>
```

This div element has position: static;

position : relative

- positioned relative to its **normal position (itself)**
- setting the top, right, bottom, and left properties affect its normal position

```
div.relative {  
    position: relative;  
    left: 100px;  
    border: 3px solid green;  
    width: 250px;  
}  
<h2>position: relative;</h2>  
<div class="relative">  
    This div element has position: relative;  
</div>
```

position: relative;

This div element has position: relative;

position : fixed

- positioned relative to the viewport
- it always stays in the same place even if the page is scrolled
- the top, right, bottom, and left properties are used to position the element

position : fixed

```
div.fixed {  
    position: fixed;  
    bottom: 0;  
    right: 0;  
    width: 300px;  
    border: 3px solid green;  
}
```

```
<div class="fixed">  
    This div element has position: fixed;  
</div>
```

This div element has position: fixed;

position : absolute

- is positioned relative to the nearest positioned ancestor
- Unlike fixed which is positioned relative to the relative to viewport.
- if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

position : absolute

```
div.relative {  
    position: relative;  
    width: 400px;  
    height: 200px;  
    border: 3px solid green;  
}
```

```
div.absolute {  
    position: absolute;  
    top: 80px;  
    right: 0;  
    width: 200px;  
    height: 100px;  
    border: 3px solid green;  
}
```

```
<div class="relative">This div element has position: relative;  
  <div class="absolute">This div element has position: absolute;</div>  
</div>
```


position : absolute

This div element has position: relative;

This div element has position:
absolute;

```
div.absolute {  
    position: absolute;  
    top: 80px;  
    right: 0;  
    width: 200px;  
    height: 100px;  
    border: 3px solid green;  
}
```

position : sticky

- It is positioned based on the user's scroll position
- A sticky element toggles between relative and fixed, depending on the scroll position.
- It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Overlapping Elements

- When elements are positioned, they can overlap other elements
- The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).
- An element can have a positive or negative stack order.

```
img {  
    position: absolute;  
    left: 0px;  
    top: 0px;  
    z-index: -1;  
}
```

CSS Layout - Overflow

What happens to content that is too big to fit into an area?

- **visible** (default)
- **hidden**
- **scroll**
- **auto**

Overflow – Visible

- overflow is visible, meaning that it is not clipped and it renders outside the element's box

```
div {  
  background-color: #eee;  
  width: 200px;  
  height: 50px;  
  border: 1px dotted black;  
  overflow: visible;  
}
```

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Overflow – Hidden

- the overflow is clipped, and the rest of the content is hidden

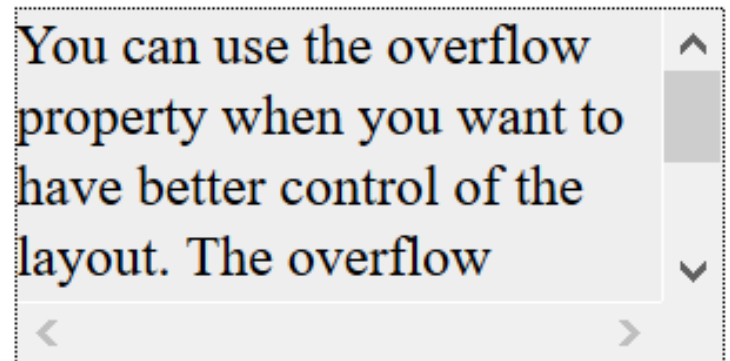
```
div {  
  background-color: #eee;  
  width: 200px;  
  height: 50px;  
  border: 1px dotted black;  
  overflow: hidden;  
}
```

You can use the overflow property when you want to have better control of the

Overflow – Scroll

- the overflow is clipped and a scrollbar is added to scroll inside the box.
- this will add a scrollbar both horizontally and vertically

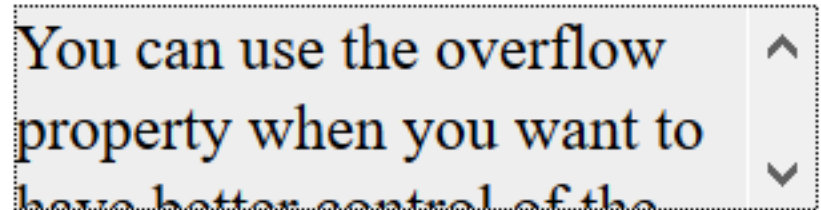
```
div {  
  background-color: #eee;  
  width: 200px;  
  height: 100px;  
  border: 1px dotted black;  
  overflow: scroll;  
}
```



Overflow – Auto

- It is similar to scroll, only it add scrollbars when necessary.

```
div {  
  background-color: #eee;  
  width: 200px;  
  height: 50px;  
  border: 1px dotted black;  
  overflow: auto;  
}
```



Overflow : Auto

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



Add a clearfix class with `overflow: auto;` to the containing element, to fix this problem:

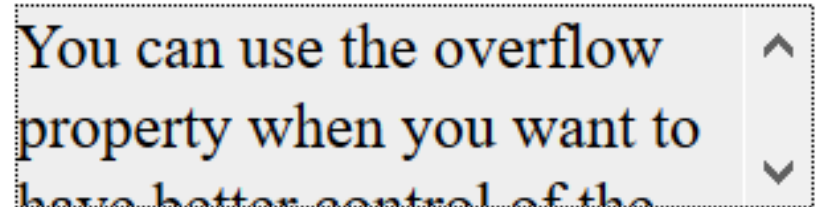
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



overflow-x and overflow-y

- The `overflow-x` and `overflow-y` properties specifies whether to change the overflow of content just horizontally or vertically (or both).

```
div {  
  background-color: #eee;  
  width: 200px;  
  height: 50px;  
  border: 1px dotted black;  
  overflow-x: hidden;  
  overflow-y: scroll;  
}
```



CSS Layout – float property

- specifies how an element should float.
- **left** - The element floats to the left of its container
- **right** - The element floats to the right of its container
- **none** - The element does not float (default)
- **inherit** - The element inherits the float value of its parent
- float property can be used to wrap text around images.

float - left

```
img {  
    float: left;  
}
```

```
<p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices



Lorem ipsum dolor sit amet, consectetur adipiscing imperdiet, nulla et dictum interdum, nisi lorem eges scelerisque enim ligula venenatis dolor. Maecenas n ultrices nec congue eget, auctor vitae massa. Fusce vestibulum augue ut aliquet. Mauris ante ligula, fac ornare eu, lobortis in odio. Praesent convallis urna interdum ut hendrerit risus congue. Nunc sagittis di ullamcorper ipsum dignissim ac. In at libero sed nui imperdiet sed ornare turpis. Donec vitae dui eget tel

float - right

```
img {  
    float: right;  
}
```

```
<p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, v scelerisque enim ligula venenatis dolor. Maecenas nisl est, ult

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida



float - none

```
img {  
    float: none;  
}
```

<p>In this example, the image will be displayed just where it occurs the text (float: none;).</p>

```
<p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices

In this example, the image will be displayed just where it occurs in the text (float: none;).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luct

The clear Property

specifies what elements can float beside the cleared element and on which side

- **none** - Allows floating elements on both sides.
(default)
- **left** - No floating elements allowed on the left side
- **right** - No floating elements allowed on the right side
- **both** - No floating elements allowed on either the left
or the right side
- **inherit** - The element inherits the clear value of
its parent

The clear Property (ex.)

```
.div1, .div3 {  
    float: left;  
    width: 100px;  
    height: 50px;  
    margin: 10px;  
    border: 3px solid #73AD21;  
}
```

```
.div2 {  
    border: 1px solid red;  
    width: 200px;  
}
```

```
.div4 {  
    border: 1px solid red;  
    width: 200px;  
    clear: left;  
}
```

<h2>Without clear</h2>

<div class="div1">div1</div>

<div class="div2">div2 </div>

<h2>With clear</h2>


<div class="div3">div3</div>

<div class="div4">div4 .</div>

CSS Layout - Horizontal & Vertical Align

Center Align Elements

```
.center {  
  margin: auto;  
  width: 60%;  
  border: 3px solid green;  
  padding: 10px;  
}
```



Center Align Elements

To horizontally center a block element (like div), use margin: auto;

Note: Using margin:auto will not work in IE8, unless a !DOCTYPE is declared.

This text is centered.

Center an Image

```
img {  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
}
```

```
<h2>Center an Image</h2>  

```

Center an Image



Left and Right Align - Using position

```
.right {  
    position: absolute;  
    right: 0px;  
    width: 300px;  
    border: 3px solid #73AD21;  
    padding: 10px;  
}
```

An example of how to right align elements with the position property:

In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.

CSS Combinators

- Specifies the relationship between the selectors
- There are four different combinators in CSS:

- | | | | |
|----|-------------------|----------|---------|
| 1) | descendant | selector | (space) |
| 2) | child | selector | (>) |
| 3) | adjacent sibling | selector | (+) |
| 4) | following sibling | selector | (~) |

Descendant Combinator (space)

- matches all elements that are descendants of a specified element.

```
div p {  
  background-color: yellow;  
  width: 200px;  
}
```

```
<div>  
  <p>Paragraph 1 in the div.</p>  
  <span><p>Paragraph 2 in the span.</p></span>  
</div>  
<p>Paragraph 3. Not in a div.</p>
```

Paragraph 1 in the div.

Paragraph 2 in the span.

Paragraph 3. Not in a div.

Child Combinator (>)

- The child selector selects all elements that are the **immediate children** of a specified element.

```
div > p {  
    background-color: yellow;  
    width: 200px;  
}
```

```
<div>  
    <p>Paragraph 1 in the div.</p>  
    <span><p>Paragraph 2 in the div.</p></span>  
</div>  
<p>Paragraph 3. Not in a div.</p>
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3. Not in a div.

General Sibling Combinator (~)

All subsequent / following siblings

```
div ~ p {  
    background-color: yellow;  
}
```

```
<p>Paragraph 1.</p>
```

```
<div>  
    <code>Some code.</code>  
    <p>Paragraph 2.</p>  
</div>
```

```
<p>Paragraph 3.</p>  
<code>Some code.</code>  
<p>Paragraph 4.</p>
```

Paragraph 1.

Some code.

Paragraph 2.

Paragraph 3.

Some code.

Paragraph 4.

CSS Pseudo-classes

- used to define a special state of an element
- It can be used to style:
 - an element when a user places a mouse over it
 - visited and unvisited links differently
 - an element when it gets focus

Examples:

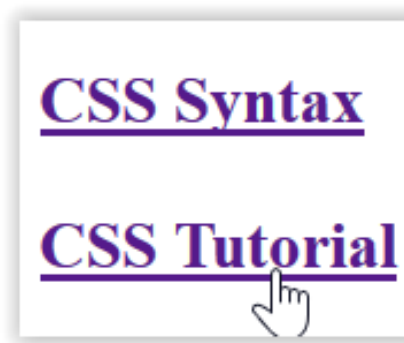
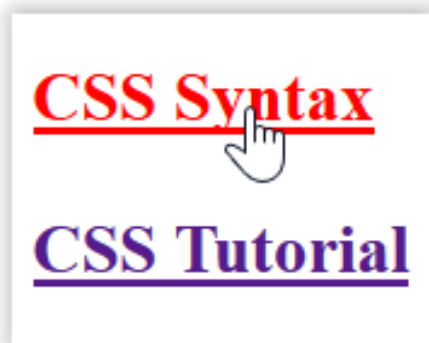
- link, visited, hover, active, nth-child, checked, focus etc.

Pseudo-classes and CSS Classes

- Pseudo-classes can be combined with CSS classes.

```
a.highlight:hover {  
    color: #ff0000;  
}
```

```
<h3><a class="highlight" href="css_syntax.asp">CSS Syntax</a></h3>  
<h3><a href="default.asp">CSS Tutorial</a></h3>
```



Pseudo-classes

- `p:first-child`
- `p span:first-child`
- `p:first-child span`

Pseudo-classes and CSS Classes

- `p:first-child`

Match all the first child `<p>` element of its parent

- `p span:first-child`

Match the first `` element in all `<p>` elements

- `p:first-child span`

Match all `` elements in all first child `<p>` element

CSS Pseudo-elements

It is used to style specified parts of an element

It can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element
- Examples
 - **first-letter, first-line, after, before, selection**

The ::first-line Pseudo-element

```
p::first-line {  
    color: red;  
    font-variant: small-caps;  
}
```

```
<p style="width:200px;">This is the sample paragraph.  
We are testing the first-line pseudo element</p>
```

THIS IS THE SAMPLE
paragraph. We are testing the
first-line pseudo element

Note: Try to **resize** the browser window and see the effect.

The `::before` Pseudo-element

```
h1::before {  
    content: url(images/smiley.jpg) ;  
}
```

```
<h1>This is a heading</h1>  
<h1>This is a heading</h1>
```



This is a heading



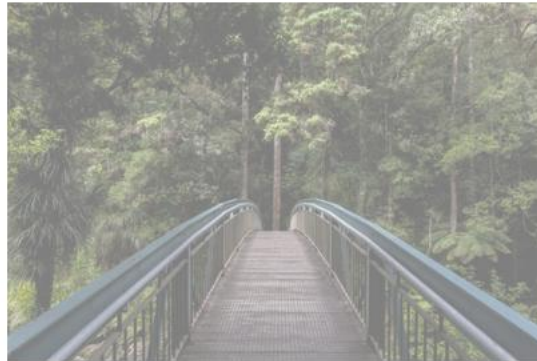
This is a heading

CSS Opacity / Transparency

- The opacity property can take a value from 0.0 – 1.0
- The lower value, the more transparent



opacity 0.2



opacity 0.5



opacity 1
(default)

CSS Attribute Selectors

- [attribute] Selector

```
a[target] {  
    background-color: yellow;  
}
```

```
<a href="https://www.ddu.ac.in">ddu.ac.in</a>
```

```
<a href="http://www.google.co.in" target="_blank">google.co.in</a>
```

```
<a href="http://www.wikipedia.org" target="_top">wikipedia.org</a>
```

[ddu.ac.in](https://www.ddu.ac.in) [google.co.in](http://www.google.co.in) [wikipedia.org](http://www.wikipedia.org)

CSS [attribute~="value"] Selector

select elements with an attribute value containing a specified word.

```
[title~="flower"] {  
    border: 5px solid yellow;  
}
```

```
  
  

```



CSS [attribute*="value"] Selector

select elements with an attribute value containing a specified word.

Note: The value does not have to be a whole word!

```
[title*=flower] {  
    border: 5px solid yellow;  
}
```

```
  
  

```



CSS [attribute|="value"] Selector

select elements with the specified attribute starting with the specified value.

```
[class|=top] {  
    background: yellow;  
    width: 200px;  
}
```

```
<h1 class="top-header">Welcome</h1>  
<p class="top-text">Hello world!</p>  
<p class="topcontent">Are you learning CSS?</p>
```

Welcome

Hello world!

Are you learning CSS?

CSS [attribute^="value"] Selector

select elements whose attribute value begins with a specified value

```
[class^="top"] {  
    background: yellow; width: 300px;  
}
```

```
<h1 class="top-header">Welcome</h1>  
<p class="top-text">Hello world!</p>  
<p class="topcontent">Are you learning CSS?</p>
```

Welcome

Hello world!

Are you learning CSS?

CSS [attribute\$="value"] Selector

select elements whose attribute value ends with a specified value.

```
[class$="test"] {  
    background: yellow;    width:300px;  
}
```

```
<div class="first_test">The first div element.</div>  
<div class="second">The second div element.</div>  
<div class="my-test">The third div element.</div>  
<p class="mytest">This is some text in a paragraph.</p>
```

The first div element.

The second div element.

The third div element.

This is some text in a paragraph.

CSS Specificity

- What if there are two or more conflicting CSS rules that point to the same element?
- Think of specificity as a score/rank that determines which style declarations are ultimately applied to an element.
- The **universal selector (*)** has low specificity, while **ID selectors** are highly specific!

Specificity Hierarchy

- 1) Inline styles (highest specificity)
- 2) IDs
- 3) Classes, attributes and pseudo-classes
- 4) Elements and pseudo-elements (lowest specificity)

Ex. 1

h1

#content h1

```
<div id="content"><h1 style="color: #ffffff">Heading</h1></div>
```

Ex. 2

- *Equal specificity: the latest rule counts*

```
h1 {background-color: yellow;}
```

```
h1 {background-color: red;}
```

```
<h1>This is heading 1</h1>
```

This is heading 1

Ex. 3

ID selectors have a higher specificity than attribute selectors

```
div#a {background-color: green;}  
#a {background-color: yellow;}  
div[id=a] {background-color: blue;}
```

```
<div id="a">This is a div</div>
```

This is a div

Ex. 4

- *A class selector beats any number of element selectors*

```
.intro {background-color: yellow;}  
h1 {background-color: red;}
```

```
<h1 class="intro">This is a heading</h1>
```

This is a heading

Specificity Rules

- Universal selector (*), combinators (+, >, ~, ' ', ||) and negation pseudo-class (:not()) have no effect on specificity.
- The selectors declared inside :not() do, however.

!important Exception

- There is a special piece of CSS that you can use to overrule all of the above calculations
- however you should be very careful with using it
- !important

!important Exception (Ex.)

```
#winning {  
    background-color: red;  
    border: 1px solid black;  
}  
.better {  
    background-color: gray;  
    border: none !important;  
}  
p {  
    background-color: blue;  
    color: white;  
    padding: 5px;  
}
```

<p class="better">This is a paragraph.</p>

<p class="better" id="winning">This is another paragraph.</p>

!important Exception (Output)

```
#winning {  
    background-color: red;  
    border: 1px solid black;  
}  
.better {  
    background-color: gray;  
    border: none !important;  
}  
p {  
    background-color: blue;  
    color: white;  
    padding: 5px;  
}
```

This is a paragraph.

This is another paragraph.

```
<p class="better">This is a paragraph.</p>  
<p class="better" id="winning">This is another paragraph.</p>
```

!important Exception (Rules of thumb)

- Always look for a way to use specificity before even considering !important
- Only use !important on page-specific CSS that overrides foreign CSS (from external libraries, like BS)
- Never use !important when you're writing a plugin
- Never use !important on site-wide CSS

References

- <https://www.w3schools.com/css/>
- <https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity>