

CSS Introduction 2

CSS Layout - The display Property

The **display** property is the most important CSS property for controlling layout.

The display Property

The **display** property specifies if/how an element is displayed.

Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is **block** or **inline**.

Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

The `<div>` element is a block-level element.

Examples of block-level elements:

- `<div>`
- `<h1>` - `<h6>`
- `<p>`
- `<form>`
- `<header>`
- `<footer>`
- `<section>`

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary.

This is an inline `` element inside a paragraph.

Examples of inline elements:

- ``
- `<a>`
- ``

Display: none;

`display: none;` is commonly used with JavaScript to hide and show elements without deleting and recreating them.

Override The Default Display Value

As mentioned, every element has a default display value. However, you can override this.

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.

A common example is making inline `` elements for horizontal menus:

Example

```
li {  
    display: inline;  
}
```

```
<head>  
<style>  
li {  
    display: inline;  
}  
</style>  
</head>  
<body>
```

```
<p>Display a list of links as a horizontal menu:</p>
```

```
<ul>  
    <li><a href="/html/default.asp" target="_blank">HTML</a></li>  
    <li><a href="/css/default.asp" target="_blank">CSS</a></li>  
    <li><a href="/js/default.asp" target="_blank">JavaScript</a></li>  
</ul>  
</body>
```

Note: Setting the display property of an element only changes **how the element is displayed**, NOT what kind of element it is. So, an inline element with `display: block;` is not allowed to have other block elements inside it.

The following example displays `` elements as block elements:

Example

```
span {
  display: block;
}

<head>
<style>
span {
  display: block;
}
</style>
</head>
<body>

<h1>Display span elements as block elements</h1>

<span>A display property with</span> <span>a value of "block" results in</span>
<span>a line break between each span elements.</span>
</body>
```

The following example displays `<a>` elements as block elements:

Example

```
a {
  display: block;
}

<head>
<style>
a {
  display: block;
}
</style>
</head>

<body>

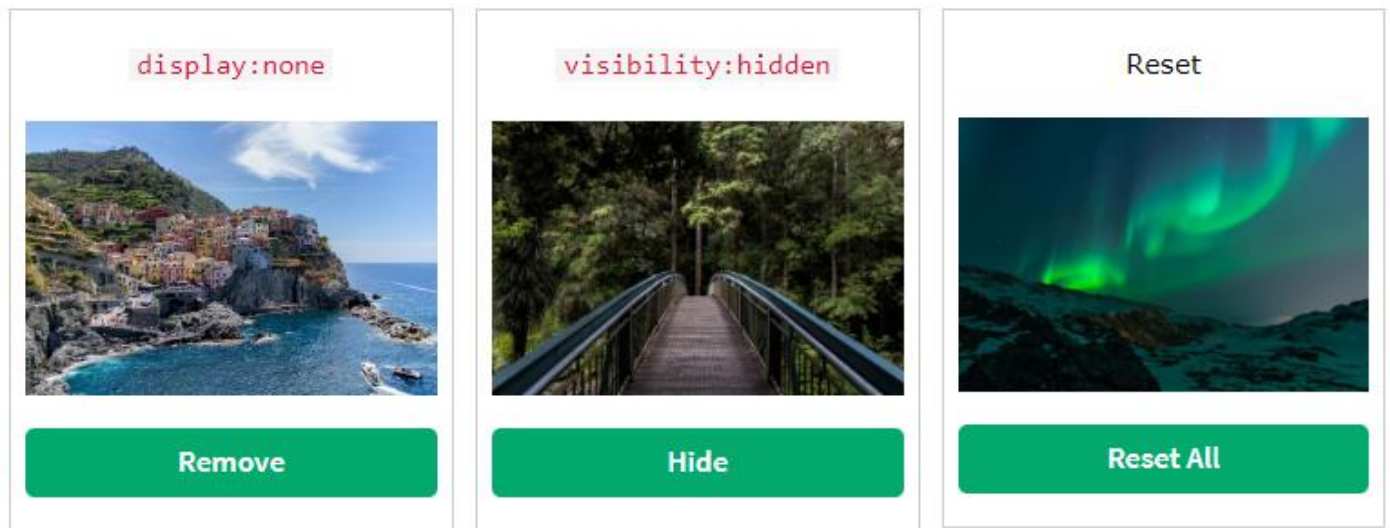
<h1>Display links as block elements</h1>

<a href="/html/default.asp" target="_blank">HTML</a>
<a href="/css/default.asp" target="_blank">CSS</a>
<a href="/js/default.asp" target="_blank">JavaScript</a>

</body>
```

Hide an Element - display:none or visibility:hidden?

Example from W3school Web Site



Hiding an element can be done by setting the **display** property to **none**. The element will be hidden, and the page will be displayed as if the element is not there:

Example

```
h1.hidden {  
  display: none;  
}
```

```
<head>  
<style>  
h1.hidden {  
  display: none;  
}  
</style>  
</head>  
<body>
```

```
<h1>This is a visible heading</h1>  
<h1 class="hidden">This is a hidden heading</h1>  
<p>Notice that the h1 element with display: none; does not take up any space.</p>  
</body>
```

visibility:hidden; also hides an element.

However, the element will still take up the same space as before. The element will be hidden, but still affect the layout:

Example

```
h1.hidden {
  visibility: hidden;
}

<head>
<style>
h1.hidden {
  visibility: hidden;
}
</style>
</head>
<body>

<h1>This is a visible heading</h1>
<h1 class="hidden">This is a hidden heading</h1>
<p>Notice that the hidden heading still takes up space.</p>
</body>
```

CSS Layout - The position Property

The **position** property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

The position Property

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

There are five different position values:

- **static**
- **relative**
- **fixed**
- **absolute**
- **sticky**

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the **position** property is set first. They also work differently depending on the position value.

position: static;

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page:

This `<div>` element has `position: static;`

Here is the CSS that is used:

Example

```
div.static {  
  position: static;  
  border: 3px solid #73AD21;  
}
```

```
<head>  
<style>  
div.static {  
  position: static;  
  border: 3px solid #73AD21;  
}  
</style>  
</head>  
<body>
```

```
<h2>position: static;</h2>
```

```
<p>An element with position: static; is not positioned in any special way; it is  
always positioned according to the normal flow of the page:</p>
```

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

position: relative;

An element with `position: relative;` is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

This `<div>` element has `position: relative;`

Here is the CSS that is used:

Example

```
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}

<head>
<style>
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>

<h2>position: relative;</h2>

<p>An element with position: relative; is positioned relative to its normal
position:</p>

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

position: fixed;

An element with **position: fixed;** is positioned relative to the viewport, which means 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.

A fixed element does not leave a gap in the page where it would normally have been located.

Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

Example

```
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
```

```
<head>
<style>
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
```

```
<h2>position: fixed;</h2>
```

<p>An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled:</p>

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

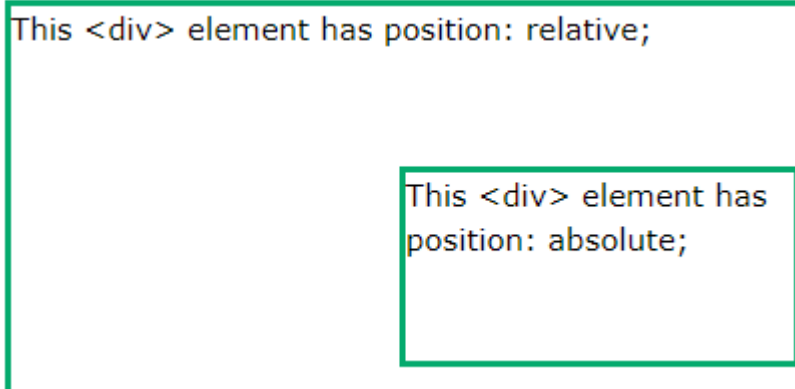
position: absolute;

An element with **position: absolute;** is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Here is a simple example:



Here is the CSS that is used:

Example

```
div.relative {  
  position: relative;  
  width: 400px;  
  height: 200px;  
  border: 3px solid #73AD21;  
}
```

```
div.absolute {  
  position: absolute;  
  top: 80px;  
  right: 0;  
  width: 200px;  
  height: 100px;  
  border: 3px solid #73AD21;  
}
```

```
<head>  
<style>  
div.relative{  
  position: relative;  
  width: 400px;  
  height: 200px;  
  border: 3px solid #73AD21;  
}
```

```
div.absolute {  
  position: absolute;  
  top: 80px;  
  right: 0;  
  width: 200px;  
  height: 100px;  
  border: 3px solid #73AD21;  
}  
</style>  
</head>  
<body>
```

```
<h2>position: absolute;</h2>
```

<p>An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):</p>

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

position: sticky;

An element with `position: sticky;` 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`).

Note: Internet Explorer does not support sticky positioning. Safari requires a `-webkit-` prefix (see example below). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (`top: 0`), when you reach its scroll position.

Example

```
div.sticky {  
  position: -webkit-sticky; /* Safari */  
  position: sticky;  
  top: 0;  
  background-color: green;  
  border: 2px solid #4CAF50;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div.sticky {  
  position: -webkit-sticky;  
  position: sticky;  
  top: 0;  
  padding: 5px;  
  background-color: #cae8ca;  
  border: 2px solid #4CAF50;  
}  
</style>  
</head>  
<body>
```

<p>Try to scroll inside this frame to understand how sticky positioning works.</p>

<div class="sticky">I am sticky!</div>

<div style="padding-bottom:2000px">
 <p>In this example, the sticky element sticks to the top of the page (top: 0), when you reach its scroll position.</p>
 <p>Scroll back up to remove the stickyness.</p>
 <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui

causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>

<p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p></div>

</body>

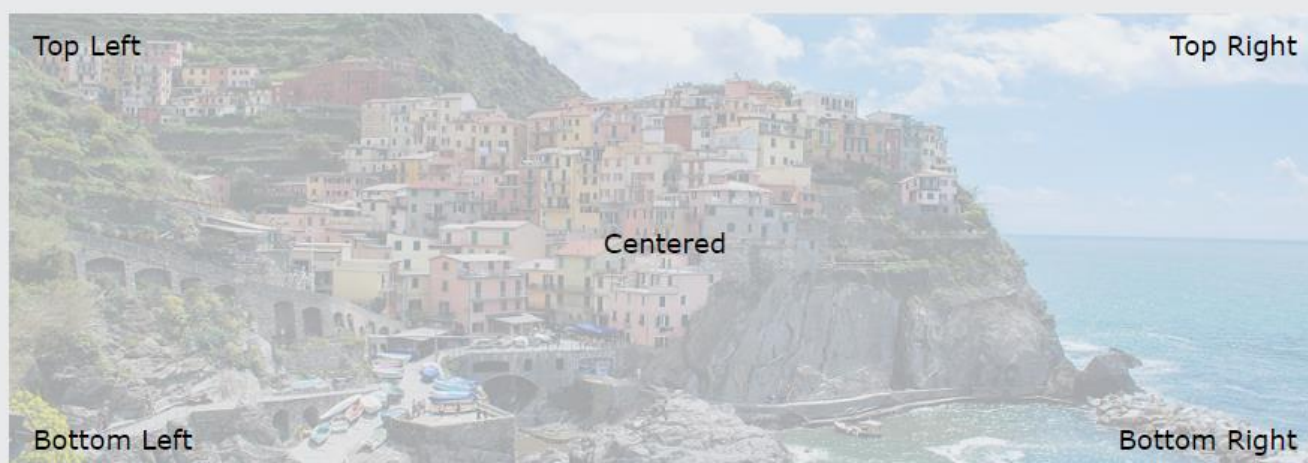
</html>

</body>

Positioning Text In an Image

How to position text over an image: (Go to w3school Example)

Example



Try it Yourself:

Top Left »

Top Right »

Bottom Left »

Bottom Right »

Centered »

CSS Layout - The z-index Property

The **z-index** property specifies the stack order of an element.

The z-index Property

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:

This is a heading

Because the image has a z-index of -1, it will be placed behind the text.



Because the image has a z-index of -1, it will be placed behind the text.

Example

```
img {  
  position: absolute;  
  left: 0px;  
  top: 0px;  
  z-index: -1;  
}  
  
<!DOCTYPE html>  
<html>  
<head>  
<style>  
img {  
  position: absolute;  
  left: 0px;  
  top: 0px;  
  z-index: -1;  
}  
</style>  
</head>  
<body>  
  
<h1>This is a heading</h1>  
  
<p>Because the image has a z-index of -1, it will be placed behind the text.</p>  
</body>  
</html>
```

Note: **z-index** only works on [positioned elements](#) (position: absolute, position: relative, position: fixed, or position: sticky) and [flex items](#) (elements that are direct children of display: flex elements).

Another z-index Example

Example

Here we see that an element with greater stack order is always above an element with a lower stack order:

```
<html>
<head>
<style>
.container {
  position: relative;
}
.black-box {
  position: relative;
  z-index: 1;
  border: 2px solid black;
  height: 100px;
  margin: 30px;
}
.gray-box {
  position: absolute;
  z-index: 3;
  background: lightgray;
  height: 60px;
  width: 70%;
  left: 50px;
  top: 50px;
}
.green-box {
  position: absolute;
  z-index: 2;
  background: lightgreen;
  width: 35%;
  left: 270px;
  top: -15px;
  height: 100px;
}
</style>
</head>
<body>
<div class="container">
  <div class="black-box">Black box</div>
  <div class="gray-box">Gray box</div>
  <div class="green-box">Green box</div>
</div>

</body>
</html>
```

Without z-index

If two positioned elements overlap each other without a **z-index** specified, the element defined **last in the HTML code** will be shown on top.

Example

Same example as above, but here with no z-index specified:

```
<html>
<head>
<style>
.container {
  position: relative;
}

.black-box {
  position: relative;
  border: 2px solid black;
  height: 100px;
  margin: 30px;
}

.gray-box {
  position: absolute;
  background: lightgray;
  height: 60px;
  width: 70%;
  left: 50px;
  top: 50px;
}

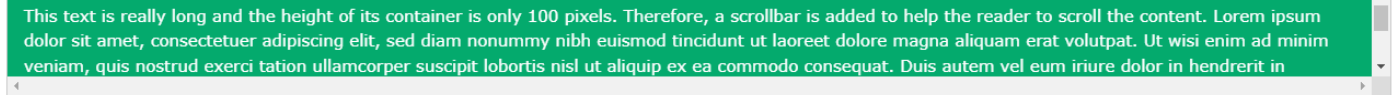
.green-box {
  position: absolute;
  background: lightgreen;
  width: 35%;
  left: 270px;
  top: -15px;
  height: 100px;
}
</style>
</head>
<body>

<div class="container">
  <div class="black-box">Black box</div>
  <div class="gray-box">Gray box</div>
  <div class="green-box">Green box</div>
</div>

</body>
</html>
```

CSS Layout - Overflow

The CSS **overflow** property controls what happens to content that is too big to fit into an area.



```
<!DOCTYPE html>
<html>
<head>
<style>
#overflowTest {
  background: #4CAF50;
  color: white;
  padding: 15px;
  width: 50%;
  height: 100px;
  overflow: scroll;
  border: 1px solid #ccc;
}
</style>
</head>
<body>
```

```
<h2>CSS Overflow</h2>
```

```
<p>The overflow property controls what happens to content that is too big to fit
into an area.</p>
```

```
<div id="overflowTest">This text is really long and the height of its container is
only 100 pixels. Therefore, a scrollbar is added to help the reader to scroll the
content. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam
nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut
wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit
lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor
in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu
feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui
blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla
facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet
doming id quod mazim placerat facer possim assum. Typi non habent claritatem
insitam; est usus legentis in iis qui facit eorum claritatem.</div>
```

```
</body>
</html>
```

CSS Overflow

The **overflow** property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The **overflow** property has the following values:

- **visible** - Default. The overflow is not clipped. The content renders outside the element's box
- **hidden** - The overflow is clipped, and the rest of the content will be invisible
- **scroll** - The overflow is clipped, and a scrollbar is added to see the rest of the content
- **auto** - Similar to **scroll**, but it adds scrollbars only when necessary

Note: The **overflow** property only works for block elements with a specified height.

Note: In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

overflow: visible

By default, the overflow is **visible**, meaning that it is not clipped and it renders outside the element's box:

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.

Example

```
div {  
  width: 200px;  
  height: 65px;  
  background-color: coral;  
  overflow: visible;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div {  
  background-color: coral;  
  width: 200px;  
  height: 65px;  
  border: 1px solid;  
  overflow: visible;  
}  
</style>  
</head>  
<body>
```

```
<h2>Overflow: visible</h2>
```

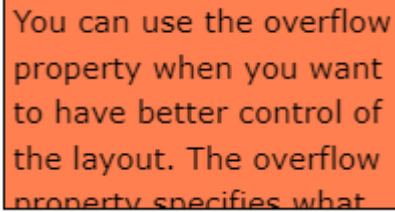
<p>By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box:</p>

<div>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.</div>

```
</body>  
</html>
```

overflow: hidden

With the `hidden` value, the overflow is clipped, and the rest of the content is hidden:



You can use the overflow
property when you want
to have better control of
the layout. The overflow
property specifies what

Example

```
div {  
  overflow: hidden;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div {  
  background-color: coral;  
  width: 200px;  
  height: 65px;  
  border: 1px solid black;  
  overflow: hidden;  
}  
</style>  
</head>  
<body>
```

```
<h2>Overflow: hidden</h2>
```

```
<p>With the hidden value, the overflow is clipped, and the rest of the content is  
hidden:</p>
```

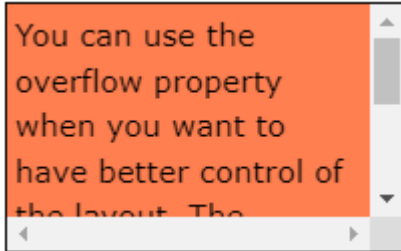
```
<p>Try to remove the overflow property to understand how it works.</p>
```

```
<div>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.</div>
```

```
</body>  
</html>
```

overflow: scroll

Setting the value to `scroll`, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):



Example

```
div {  
  overflow: scroll;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div {  
  background-color: coral;  
  width: 200px;  
  height: 100px;  
  border: 1px solid black;  
  overflow: scroll;  
}  
</style>  
</head>  
<body>
```

```
<h2>Overflow: scroll</h2>
```

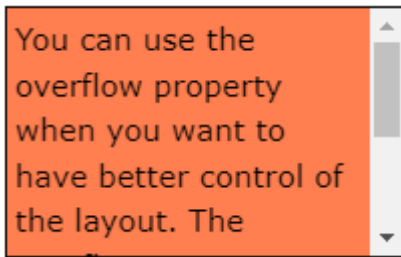
<p>Setting the overflow value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):</p>

<div>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.</div>

```
</body>  
</html>
```

overflow: auto

The `auto` value is similar to `scroll`, but it adds scrollbars only when necessary:



Example

```
div {  
  overflow: auto;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div {  
  background-color: coral;  
  width: 200px;  
  height: 65px;  
  border: 1px solid black;  
  overflow: auto;  
}  
</style>  
</head>  
<body>
```

```
<h2>Overflow: auto</h2>
```

```
<p>The auto value is similar to scroll, only it add scrollbars when necessary:</p>
```

```
<div>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.</div>
```

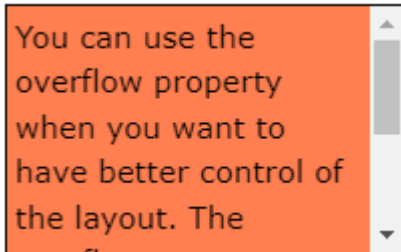
```
</body>  
</html>
```

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):

`overflow-x` specifies what to do with the left/right edges of the content.

`overflow-y` specifies what to do with the top/bottom edges of the content.



Example

```
div {  
  overflow-x: hidden; /* Hide horizontal scrollbar */  
  overflow-y: scroll; /* Add vertical scrollbar */  
}  
  
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div {  
  background-color: coral;  
  width: 200px;  
  height: 65px;  
  border: 1px solid black;  
  overflow-x: hidden;  
  overflow-y: scroll;  
}  
</style>  
</head>  
<body>  
  
<h2>Overflow-x and overflow-y</h2>  
  
<p>You can also change the overflow of content horizontally or vertically.</p>  
<p>overflow-x specifies what to do with the left/right edges of the content.</p>  
<p>overflow-y specifies what to do with the top/bottom edges of the content.</p>  
  
<div>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.</div>  
  
</body>  
</html>
```

CSS Layout - float and clear

The CSS `float` property specifies how an element should float.

The CSS `clear` property specifies what elements can float beside the cleared element and on which side.



Float Left



Float Right

The float Property

The `float` property is used for positioning and formatting content e.g. let an image float left to the text in a container.

The `float` property can have one of the following values:

- `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 (will be displayed just where it occurs in the text). This is default
- `inherit` - The element inherits the float value of its parent

In its simplest use, the `float` property can be used to wrap text around images.

Example - float: right;

The following example specifies that an image should float to the **right** in a text:

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...



```
<!DOCTYPE html>
<html>
<head>
<style>
img {
  float: right;
}
</style>
</head>
<body>
```

```
<h2>Float Right</h2>
```

```
<p>In this example, the image will float to the right in the paragraph, and the
text in the paragraph will wrap around the image.</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 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 venenatis. Integer fringilla congue eros non fermentum. Sed dapibus
pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>

</body>
</html>
```

Example - float: left;

The following example specifies that an image should float to the **left** in a text:



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...

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
  float: left;
}
</style>
</head>
<body>
```

```
<h2>Float Left</h2>
```

<p>In this example, the image will float to the left in the paragraph, and the text in the paragraph will wrap around the image.</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 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 venenatis. Integer fringilla congue eros non fermentum. Sed dapibus
pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>
```

```
</body>
</html>
```


Example - No float

In the following 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 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...

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

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

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Float None</h2>
```

```
<p>In this example, the image will be displayed just where it occurs in 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 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 venenatis. Integer fringilla congue eros non fermentum. Sed dapib  
pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>
```

```
</body>
```

```
</html>
```

Example - Float Next To Each Other

Normally div elements will be displayed on top of each other. However, if we use `float: left` we can let elements float next to each other:

Example

```
div {
  float: left;
  padding: 15px;
}

.div1 {
  background: red;
}

.div2 {
  background: yellow;
}

.div3 {
  background: green;
}

<!DOCTYPE html>
<html>
<head>
<style>
div {
  float: left;
  padding: 15px;
}
.div1 {
  background: red;
}

.div2 {
  background: yellow;
}
.div3 {
  background: green;
}
</style>
</head>
<body>
<h2>Float Next To Each Other</h2>
<p>In this example, the three divs will float next to each other.</p>
<div class="div1">Div 1</div>
<div class="div2">Div 2</div>
<div class="div3">Div 3</div>

</body>
</html>
```

CSS Layout - display: inline-block

The display: inline-block Value

Compared to `display: inline`, the major difference is that `display: inline-block` allows to set a width and height on the element.

Also, with `display: inline-block`, the top and bottom margins/paddings are respected, but with `display: inline` they are not.

Compared to `display: block`, the major difference is that `display: inline-block` does not add a line-break after the element, so the element can sit next to other elements.

The following example shows the different behavior of `display: inline`, `display: inline-block` and `display: block`:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
span.a {
  display: inline; /* the default for span */
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
span.b {
  display: inline-block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
span.c {
  display: block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
</style>
```

```

</head>

<body>

<h1>The display Property</h1>

<h2>display: inline</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat
scelerisque elit sit amet consequat. Aliquam erat volutpat. <span
class="a">Aliquam</span> <span class="a">venenatis</span> gravida nisl sit amet
facilisis. Nullam cursus fermentum velit sed laoreet. </div>

<h2>display: inline-block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat
scelerisque elit sit amet consequat. Aliquam erat volutpat. <span
class="b">Aliquam</span> <span class="b">venenatis</span> gravida nisl sit amet
facilisis. Nullam cursus fermentum velit sed laoreet. </div>

<h2>display: block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat
scelerisque elit sit amet consequat. Aliquam erat volutpat. <span
class="c">Aliquam</span> <span class="c">venenatis</span> gravida nisl sit amet
facilisis. Nullam cursus fermentum velit sed laoreet. </div>

</body>
</html>

```

Using inline-block to Create Navigation Links

One common use for `display: inline-block` is to display list items horizontally instead of vertically. The following example creates horizontal navigation links:

Example

```

<!DOCTYPE html>
<html>
<head>
<style>
.nav {
  background-color: yellow;
  list-style-type: none;
  text-align: center;
  padding: 0;
  margin: 0;
}
.nav li {
  display: inline-block;
  font-size: 20px;
  padding: 20px;
}
</style>
</head>

```

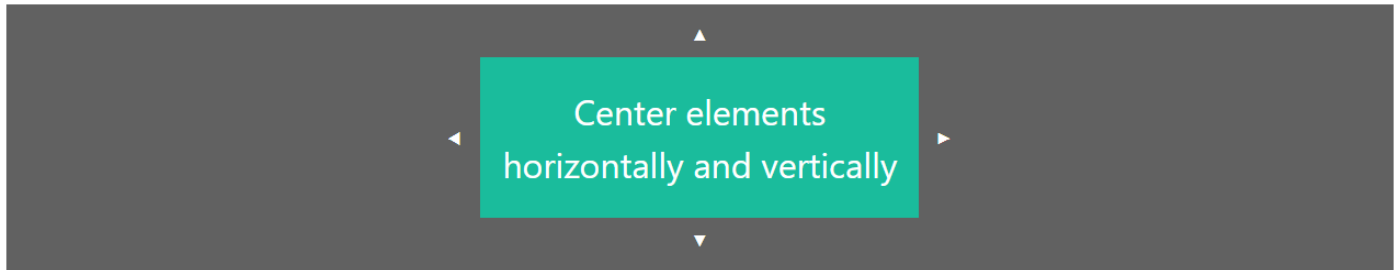
```
<body>

<h1>Horizontal Navigation Links</h1>
<p>By default, list items are displayed vertically. In this example we use display:
inline-block to display them horizontally (side by side).</p>
<p>Note: If you resize the browser window, the links will automatically break when it
becomes too crowded.</p>

<ul class="nav">
  <li><a href="#home">Home</a></li>
  <li><a href="#about">About Us</a></li>
  <li><a href="#clients">Our Clients</a></li>
  <li><a href="#contact">Contact Us</a></li>
</ul>

</body>
</html>
```

CSS Layout - Horizontal & Vertical Align



Center Align Elements

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

Setting the width of the element will prevent it from stretching out to the edges of its container.

The element will then take up the specified width, and the remaining space will be split equally between the two margins:



This div element is centered.

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  margin: auto;
  width: 50%;
  border: 3px solid green;
  padding: 10px;
}
</style>
</head>
<body>
<h2>Center Align Elements</h2>
<p>To horizontally center a block element (like div), use margin: auto;</p>
<div class="center">
  <p>Hello World!</p>
</div>

</body>
</html>
```

Note: Center aligning has no effect if the `width` property is not set (or set to 100%).

Center Align Text

To just center the text inside an element, use `text-align: center;`

This text is centered.

Example

```
.center {
  text-align: center;
  border: 3px solid green;
}

<!DOCTYPE html>
<html>
<head>
<style>
.center {
  text-align: center;
  border: 3px solid green;
}
</style>
</head>
<body>

<h2>Center Text</h2>

<div class="center">
  <p>This text is centered.</p>
</div>

</body>
</html>
```

Tip: For more examples on how to align text, see the [CSS Text](#) chapter.

Center an Image

To center an image, set left and right margin to **auto** and make it into a **block** element:



Example

```
img {  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
  width: 40%;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
img {  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
}  
</style>  
</head>  
<body>
```

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

```
<p>To center an image, set left and right margin to auto, and make it into a block  
element.</p>
```

```

```

```
</body>  
</html>
```


Left and Right Align - Using position

One method for aligning elements is to use `position: absolute;`

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

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

Example

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

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  .right {  
    position: absolute;  
    right: 0px;  
    width: 300px;  
    border: 3px solid #73AD21;  
    padding: 10px;  
  }  
</style>  
</head>  
<body>
```

```
<h2>Right align with the position property</h2>
```

```
<p>An example of how to right align elements with the position property:</p>
```

```
<div class="right">  
  <p>In my younger and more vulnerable years my father gave me some advice that I've  
  been turning over in my mind ever since.</p>  
</div>  
  
</body>  
</html>
```

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Left and Right Align - Using float

Another method for aligning elements is to use the `float` 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.

Example

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

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

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Right align with the position property</h2>
```

```
<p>An example of how to right align elements with the position property:</p>
```

```
<div class="right">
```

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

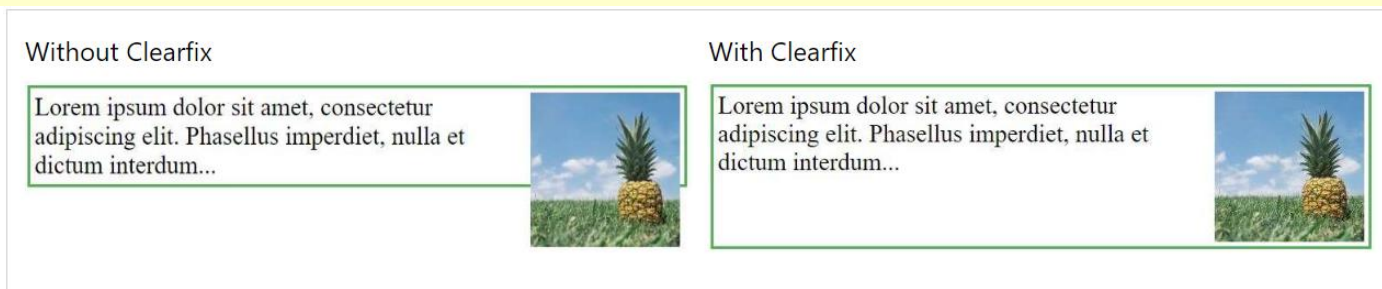
```
</div>
```

```
</body>
```

```
</html>
```

The clearfix Hack

Note: If an element is taller than the element containing it, and it is floated, it will overflow outside of its container. You can use the "clearfix hack" to fix this (see example below).



Then we can add the clearfix hack to the containing element to fix this problem:

Example

```
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}

<!DOCTYPE html>
<html>
<head>
<style>
div {
  border: 3px solid #4CAF50;
  padding: 5px;
}

.img1 {
  float: right;
}

.img2 {
  float: right;
}

.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
</style>
</head>
<body>

<h2>Without Clearfix</h2>
```

<p>This image is floated to the right. It is also taller than the element containing it, so it overflows outside of its container:</p>

```
<div>
  
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet...
</div>
```

<h2 style="clear:right">With New Modern Clearfix</h2>

<p>Add the clearfix hack to the containing element, to fix this problem:</p>


```
<div class="clearfix">
  
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet...
</div>
```

</body>

</html>

Center Vertically - Using padding

There are many ways to center an element vertically in CSS. A simple solution is to use top and bottom **padding**:



I am vertically centered.

I am vertically centered.

Example

```
.center {  
  padding: 70px 0;  
  border: 3px solid green;  
}
```

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  .center {  
    padding: 70px 0;  
    border: 3px solid green;  
  }  
</style>  
</head>  
<body>
```

```
<h2>Center vertically with padding</h2>
```

```
<p>In this example, we use the padding property to center the div element  
vertically:</p>
```

```
<div class="center">  
  <p>I am vertically centered.</p>  
</div>  
  
</body>  
</html>
```

To center both vertically and horizontally, use `padding` and `text-align: center`:

I am vertically and horizontally centered.

Example

```
.center {  
  padding: 70px 0;  
  border: 3px solid green;  
  text-align: center;  
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
.center {  
  padding: 70px 0;  
  border: 3px solid green;  
  text-align: center;  
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Center with padding and text-align</h2>
```

```
<p>In this example, we use padding and text-align to center the div element both  
vertically and horizontally:</p>
```

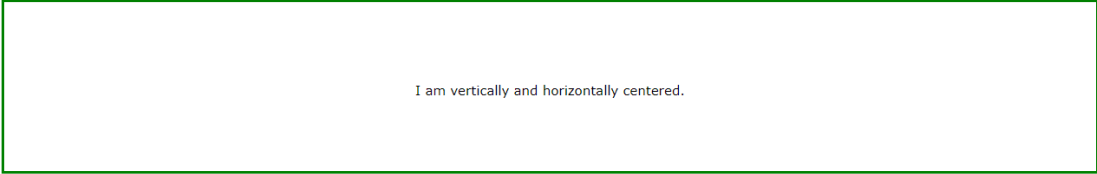
```
<div class="center">  
  <p>I am vertically and horizontally centered.</p>  
</div>
```

```
</body>
```

```
</html>
```

Center Vertically - Using line-height

Another trick is to use the `line-height` property with a value that is equal to the `height` property:



I am vertically and horizontally centered.

Example

```
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}
/* If the text has multiple lines, add the following: */
.center p {
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}
.center p {
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
</style>
</head>
<body>
<h2>Center with line-height</h2>
<p>In this example, we use the line-height property with a value that is equal to the
height property to center the div element:</p>
<div class="center">
  <p>I am vertically and horizontally centered.</p>
</div>
</body>
</html>
```

Center Vertically - Using position & transform

If **padding** and **line-height** are not options, another solution is to use positioning and the **transform** property:

I am vertically and horizontally centered.

Example

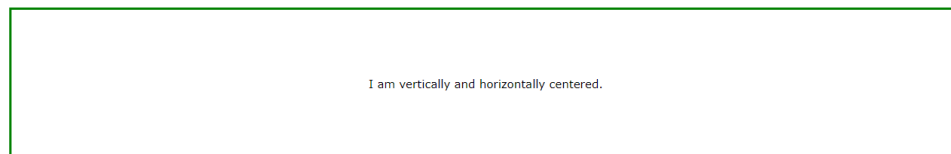
```
.center {  
  height: 200px;  
  position: relative;  
  border: 3px solid green;  
}  
  
.center p {  
  margin: 0;  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
}  
  
<!DOCTYPE html>  
<html>  
<head>  
<style>  
  .center {  
    height: 200px;  
    position: relative;  
    border: 3px solid green;  
  }  
  
  .center p {  
    margin: 0;  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    -ms-transform: translate(-50%, -50%);  
    transform: translate(-50%, -50%);  
  }  
</style>  
</head>  
<body>  
  
<h2>Center with position and transform</h2>
```


<p>In this example, we use positioning and the transform property to vertically and horizontally center the div element:</p>

```
<div class="center">
  <p>I am vertically and horizontally centered.</p>
</div>
</body>
</html>
```

Center Vertically - Using Flexbox

You can also use flexbox to center things. Just note that flexbox is not supported in IE10 and earlier versions:



I am vertically and horizontally centered.

Example

```
.center {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 200px;
  border: 3px solid green;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 200px;
  border: 3px solid green;
}
</style>
</head>
<body>
<h2>Flexbox Centering</h2>
<p>A container with both the justify-content and the align-items properties set to
<em>center</em> will align the item(s) in the center (in both axis).</p>
<div class="center">
  <p>I am vertically and horizontally centered.</p>
</div>
</body>
</html>
```

CSS Combinators

CSS Combinators

A combinator is something that explains the relationship between the selectors.

A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.

There are four different combinators in CSS:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)

Descendant Selector

The descendant selector matches all elements that are descendants of a specified element.

The following example selects all <p> elements inside <div> elements:

Example

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

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div p {  
  background-color: yellow;  
}  
</style>  
</head>  
<body>
```

<h2>Descendant Selector</h2>

<p>The descendant selector matches all elements that are descendants of a specified element.</p>

```
<div>  
  <p>Paragraph 1 in the div.</p>  
  <p>Paragraph 2 in the div.</p>
```

```

    <section><p>Paragraph 3 in the div.</p></section>
</div>

<p>Paragraph 4. Not in a div.</p>
<p>Paragraph 5. Not in a div.</p>

</body>
</html>

```

Child Selector (>)

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

The following example selects all <p> elements that are children of a <div> element:

Example

```

div > p {
    background-color: yellow;
}

```

```

<!DOCTYPE html>
<html>
<head>
<style>
div > p {
    background-color: yellow;
}
</style>
</head>
<body>

```

```
<h2>Child Selector</h2>
```

<p>The child selector (>) selects all elements that are the children of a specified element.</p>

```

<div>
    <p>Paragraph 1 in the div.</p>
    <p>Paragraph 2 in the div.</p>
    <section>
        <!-- not Child but Descendant -->
        <p>Paragraph 3 in the div (inside a section element).</p>
    </section>
    <p>Paragraph 4 in the div.</p>
</div>
<p>Paragraph 5. Not in a div.</p>
<p>Paragraph 6. Not in a div.</p>

</body>
</html>

```

Adjacent Sibling Selector (+)

The adjacent sibling selector is used to select an element that is directly after another specific element.

Sibling elements must have the same parent element, and "adjacent" means "immediately following".

The following example selects the first <p> element that are placed immediately after <div> elements:

Example

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
div + p {
```

```
    background-color: yellow;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Adjacent Sibling Selector</h2>
```

```
<p>The + selector is used to select an element that is directly after another specific element.</p>
```

```
<p>The following example selects the first p element that are placed immediately after div elements:</p>
```

```
<div>
```

```
    <p>Paragraph 1 in the div.</p>
```

```
    <p>Paragraph 2 in the div.</p>
```

```
</div>
```

```
<p>Paragraph 3. After a div.</p>
```

```
<p>Paragraph 4. After a div.</p>
```

```
<div>
```

```
    <p>Paragraph 5 in the div.</p>
```

```
    <p>Paragraph 6 in the div.</p>
```

```
</div>
```

```
<p>Paragraph 7. After a div.</p>
```

```
<p>Paragraph 8. After a div.</p>
```

```
</body>
```

```
</html>
```

General Sibling Selector (~)

The general sibling selector selects all elements that are next siblings of a specified element.

The following example selects all <p> elements that are next siblings of <div> elements:

Example

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

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
div ~ p {  
  background-color: yellow;  
}  
</style>  
</head>  
<body>
```

```
<h2>General Sibling Selector</h2>
```

```
<p>The general sibling selector (~) selects all elements that are next siblings of a  
specified element.</p>
```

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

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

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

```
</body>  
</html>
```

CSS Pseudo-classes

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user mouses over it
- Style visited and unvisited links differently
- Style an element when it gets focus

Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {  
  property: value;  
}
```

Anchor Pseudo-classes

Links can be displayed in different ways:

Example

```
/* unvisited link */  
a:link {  
  color: #FF0000;  
}  
  
/* visited link */  
a:visited {  
  color: #00FF00;  
}  
  
/* mouse over link */  
a:hover {  
  color: #FF00FF;  
}  
  
/* selected link */  
a:active {  
  color: #0000FF;  
}
```

```

<!DOCTYPE html>
<html>
<head>
<style>
/* unvisited link */
a:link {
    color: red;
}

/* visited link */
a:visited {
    color: green;
}

/* mouse over link */
a:hover {
    color: hotpink;
}

/* selected link */
a:active {
    color: blue;
}
</style>
</head>
<body>

<h2>Styling a link depending on state</h2>

<p><b><a href="default.asp" target="_blank">This is a link</a></b></p>
<p><b>Note:</b> a:hover MUST come after a:link and a:visited in the CSS definition in
order to be effective.</p>
<p><b>Note:</b> a:active MUST come after a:hover in the CSS definition in order to be
effective.</p>

</body>
</html>

```

Note: `a:hover` MUST come after `a:link` and `a:visited` in the CSS definition in order to be effective! `a:active` MUST come after `a:hover` in the CSS definition in order to be effective! Pseudo-class names are not case-sensitive.

Pseudo-classes and HTML Classes

Pseudo-classes can be combined with HTML classes:

When you hover over the link in the example, it will change color:

Example

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

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
a.highlight:hover {  
    color: #ff0000;  
    font-size: 22px;  
}  
</style>  
</head>  
<body>
```

```
<h2>Pseudo-classes and HTML Classes</h2>
```

```
<p>When you hover over the first link below, it will change color and font size:</p>
```

```
<p><a class="highlight" href="css_syntax.asp">CSS Syntax</a></p>
```

```
<p><a href="default.asp">CSS Tutorial</a></p>
```

```
</body>  
</html>
```


Hover on <div>

An example of using the `:hover` pseudo-class on a `<div>` element:

Example

```
div:hover {
  background-color: blue;
}
<!DOCTYPE html>
<html>
<head>
<style>
div {
  background-color: green;
  color: white;
  padding: 25px;
  text-align: center;
}

div:hover {
  background-color: blue;
}
</style>
</head>
<body>

<p>Mouse over the div element below to change its background color:</p>

<div>Mouse Over Me</div>

</body>
</html>
```

Simple Tooltip Hover

Hover over a <div> element to show a <p> element (like a tooltip):

Hover over me to show the <p> element.

Example

```
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}

div:hover p {
  display: block;
}

<!DOCTYPE html>
<html>
<head>
<style>
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}

div:hover p {
  display: block;
}
</style>
</head>
<body>

<div>Hover over this div element to show the p element
  <p>Tada! Here I am!</p>
</div>

</body>
</html>
```

CSS - The :first-child Pseudo-class

The `:first-child` pseudo-class matches a specified element that is the first child of another element.

Match the first <p> element

In the following example, the selector matches any <p> element that is the first child of any element:

Example

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

<!DOCTYPE html>
<html>
<head>
<style>
p:first-child {
  color: blue;
}
</style>
</head>
<body>

<p>This is some text.</p>
<p>This is some text.</p>

<div>
  <p>This is some text.</p>
  <p>This is some text.</p>
</div>

</body>
</html>
```

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

In the following example, the selector matches the first `<i>` element in all `<p>` elements:

Example

```
p i:first-child {
  color: blue;
}

<!DOCTYPE html>
<html>
<head>
<style>
p i:first-child {
  color: blue;
}
</style>
</head>
<body>

<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>

</body>
</html>
```

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

In the following example, the selector matches all `<i>` elements in `<p>` elements that are the first child of another element:

Example

```
p:first-child i {
  color: blue;
}

<!DOCTYPE html>
<html>
<head>
<style>
p:first-child i {
  color: blue;
}
</style>
</head>
<body>

<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
```

```
<div>
  <p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
  <p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
</div>

</body>
</html>
```

CSS - The :lang Pseudo-class

The `:lang` pseudo-class allows you to define special rules for different languages.

In the example below, `:lang` defines the quotation marks for `<q>` elements with `lang="no"`:

Example

```
<html>
<head>
<style>
q:lang(no) {
  quotes: "~" "~";
}
</style>
</head>
<body>

<p>Some text <q lang="no">A quote in a paragraph</q> Some text.</p>

</body>
</html>
```

What are Pseudo-Elements?

For example, it can be used to:

- # Syntax

```
selector::pseudo-element {
  property: value;
}
```

The `::first-line` pseudo-element is used to add a special style to the first line of a text.

Example

```
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
</style>
</head>
<body>
```

54

```
</body>
</html>
```

Note: The `::first-line` pseudo-element can only be applied to block-level elements.

The following properties apply to the `::first-line` pseudo-element:

- font properties
- color properties
- background properties
- word-spacing
- letter-spacing
- text-decoration
- vertical-align
- text-transform
- line-height
- clear

Notice the double colon notation - `::first-line` versus `:first-line`

The double colon replaced the single-colon notation for pseudo-elements in CSS3. This was an attempt from W3C to distinguish between **pseudo-classes** and **pseudo-elements**.

The single-colon syntax was used for both pseudo-classes and pseudo-elements in CSS2 and CSS1.

For backward compatibility, the single-colon syntax is acceptable for CSS2 and CSS1 pseudo-elements.

The `::first-letter` Pseudo-element

The `::first-letter` pseudo-element is used to add a special style to the first letter of a text.

The following example formats the first letter of the text in all `<p>` elements:

Example

```
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
p::first-letter {
  color: #ff0000;
```

```
    font-size: xx-large;
}
</style>
</head>
<body>
```

<p>You can use the `::first-letter` pseudo-element to add a special effect to the first character of a text!</p>

```
</body>
</html>
```

Note: The `::first-letter` pseudo-element can only be applied to block-level elements.

The following properties apply to the `::first-letter` pseudo- element:

- font properties
- color properties
- background properties
- margin properties
- padding properties
- border properties
- text-decoration
- vertical-align (only if "float" is "none")
- text-transform
- line-height
- float
- clear

Pseudo-elements and HTML Classes

Pseudo-elements can be combined with HTML classes:

Example

```
p.intro::first-letter {
    color: #ff0000;
    font-size: 200%;
}
<!DOCTYPE html>
<html>
<head>
<style>
p.intro::first-letter {
    color: #ff0000;
    font-size: 200%;
}
</style>
</head>
<body>
```

```
<p class="intro">This is an introduction.</p>
<p>This is a paragraph with some text. A bit more text even.</p>
```



```
</body>
</html>
```

The example above will display the first letter of paragraphs with class="intro", in red and in a larger size.

Multiple Pseudo-elements

Several pseudo-elements can also be combined.

In the following example, the first letter of a paragraph will be red, in an xx-large font size. The rest of the first line will be blue, and in small-caps. The rest of the paragraph will be the default font size and color:

Example

```
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}
```

```
p::first-line {
  color: #0000ff;
  font-variant: small-caps;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}

p::first-line {
  color: #0000ff;
  font-variant: small-caps;
}
</style>
</head>
<body>
```

<p>You can combine the ::first-letter and ::first-line pseudo-elements to add a special effect to the first letter and the first line of a text!</p>

```
</body>
</html>
```

CSS - The ::before Pseudo-element

The **::before** pseudo-element can be used to insert some content before the content of an element.

The following example inserts an image before the content of each `<h1>` element:

Example

```
h1::before {
  content: url(smiley.gif);
}

<!DOCTYPE html>
<html>
<head>
<style>
h1::before {
  content: url(smiley.gif);
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>The ::before pseudo-element inserts content before the content of an element.</p>

<h1>This is a heading</h1>

</body>
</html>
```

CSS - The ::after Pseudo-element

The **::after** pseudo-element can be used to insert some content after the content of an element.

The following example inserts an image after the content of each `<h1>` element:

Example

```
h1::after {
  content: url(smiley.gif);
}
```

```

<!DOCTYPE html>
<html>
<head>
<style>
h1::after {
  content: url(smiley.gif);
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>The ::after pseudo-element inserts content after the content of an element.</p>

<h1>This is a heading</h1>

</body>
</html>

```

CSS - The ::marker Pseudo-element

The `::marker` pseudo-element selects the markers of list items.

The following example styles the markers of list items:

Example

```

::marker {
  color: red;
  font-size: 23px;
}
<!DOCTYPE html>
<html>
<head>
<style>
::marker {
  color: red;
  font-size: 23px;
}
</style>
</head>
<body>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
<ol>
  <li>First</li>
  <li>Second</li>
  <li>Third</li>
</ol>
</body>
</html>

```

CSS - The ::selection Pseudo-element

The `::selection` pseudo-element matches the portion of an element that is selected by a user.

The following CSS properties can be applied to `::selection`: `color`, `background`, `cursor`, and `outline`.

The following example makes the selected text red on a yellow background:

Example

```
::selection {
  color: red;
  background: yellow;
}

<!DOCTYPE html>
<html>
<head>
<style>
::selection {
  color: red;
  background: yellow;
}
</style>
</head>
<body>

<h1>Select some text on this page:</h1>

<p>This is a paragraph.</p>
<div>This is some text in a div element.</div>

</body>
</html>
```