**What is CSS?**

CSS stands for Cascading Style Sheets

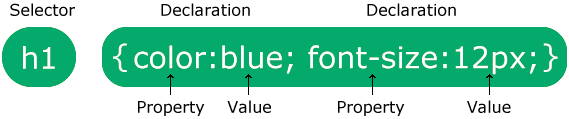
CSS describes how HTML elements are to be displayed on screen, paper, or in other media

CSS saves a lot of work. It can control the layout of multiple web pages all at once

External stylesheets are stored in CSS files

**CSS Syntax**

A CSS rule consists of a selector and a declaration block.



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

*Example*

In this example all <p> elements will be center-aligned, with a red text color:

p {

color: red;

text-align: center;

}

p is a selector in CSS (it points to the HTML element you want to style: <p>).

color is a property, and red is the property value

text-align is a property, and center is the property value

**CSS Selectors**

**CSS selectors are used to "find" (or select) the HTML elements you want to style.**

**We can divide CSS selectors into five categories:**

**Simple selectors (select elements based on name, id, class)**

Combinator selectors (select elements based on a specific relationship between them)

Pseudo-class selectors (select elements based on a certain state)

Pseudo-elements selectors (select and style a part of an element)

Attribute selectors (select elements based on an attribute or attribute value)

This page will explain the most basic CSS selectors.

**The CSS element Selector**

The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

p {

text-align: center;

color: red;

}

**The CSS id Selector**

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

#para1 {

text-align: center;

color: red;

}

Note: An id name cannot start with a number!

**The CSS class Selector**

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

Example

In this example all HTML elements with class="center" will be red and center-aligned:

.center {

text-align: center;

color: red;

}

You can also specify that only specific HTML elements should be affected by a class.

Example

In this example only <p> elements with class="center" will be red and center-aligned:

p.center {

text-align: center;

color: red;

}

HTML elements can also refer to more than one class.

Example

In this example the <p> element will be styled according to class="center" and to class="large":

<p class="center large">This paragraph refers to two classes.</p>

Note: A class name cannot start with a number!

<head>

<style>

p.center {

text-align: center;

color: red;

}

p.large {

font-size: 300%;

}

</style>

</head>

<body>

<h1 class="center">This heading will not be affected</h1>

<p class="center">This paragraph will be red and center-aligned.</p>

<p class="center large">This paragraph will be red, center-aligned, and in a large font-size.</p>

</body>

**The CSS Universal Selector**

The universal selector (\*) selects all HTML elements on the page.

Example

The CSS rule below will affect every HTML element on the page:

\* {

text-align: center;

color: blue;

}

**The CSS Grouping Selector**

The grouping selector selects all the HTML elements with the same style definitions.

Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

h1 {

text-align: center;

color: red;

}

h2 {

text-align: center;

color: red;

}

p {

text-align: center;

color: red;

}

It will be better to group the selectors, to minimize the code.

To group selectors, separate each selector with a comma.

Example

In this example we have grouped the selectors from the code above:

h1, h2, p {

text-align: center;

color: red;

}

**Three Ways to Insert CSS**

There are three ways of inserting a style sheet:

* External CSS
* Internal CSS
* Inline CSS

**External CSS**

With an external style sheet, you can change the look of an entire website by changing just one file!

Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

Example

External styles are defined within the <link> element, inside the <head> section of an HTML page:

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="mystyle.css">

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

An external style sheet can be written in any text editor, and must be saved with a .css extension.

The external .css file should not contain any HTML tags.

Here is how the "mystyle.css" file looks:

"mystyle.css"

body {

background-color: lightblue;

}

h1 {

color: navy;

margin-left: 20px;

}

Note: Do not add a space between the property value (20) and the unit (px):

Incorrect (space): margin-left: 20 px;

Correct (no space): margin-left: 20px;

**Internal CSS**

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

Example

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-color: linen;

}

h1 {

color: maroon;

margin-left: 40px;

}

</style>

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

**Inline CSS**

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

Example

Inline styles are defined within the "style" attribute of the relevant element:

<!DOCTYPE html>

<html>

<body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>

<p style="color:red;">This is a paragraph.</p>

</body>

</html>

**Multiple Style Sheets**

If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used.

Assume that an external style sheet has the following style for the <h1> element:

h1 {

color: navy;

}

Then, assume that an internal style sheet also has the following style for the <h1> element:

h1 {

color: orange;

}

Example

If the internal style is defined after the link to the external style sheet, the <h1> elements will be "orange":

<head>

<link rel="stylesheet" type="text/css" href="mystyle.css">

<style>

h1 {

color: orange;

}

</style>

</head>

Example

However, if the internal style is defined before the link to the external style sheet, the <h1> elements will be "navy":

<head>

<style>

h1 {

color: orange;

}

</style>

<link rel="stylesheet" type="text/css" href="mystyle.css">

</head>

**Cascading Order**

What style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

Inline style (inside an HTML element)

External and internal style sheets (in the head section)

Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

**CSS Comments**

A CSS comment is placed inside the <style> element, and starts with /\* and ends with \*/:

p {

color: red; /\* Set text color to red \*/

}

**CSS Colors**

CSS RGB Colors

In CSS, a color can be specified as an RGB value, using this formula:

**rgb(red, green, blue)**

Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.

For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.

To display black, set all color parameters to 0, like this: rgb(0, 0, 0).

To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

Experiment by mixing the RGB values below:

## **RGBA Value**

An RGBA color value is specified with:

**rgba(*red,* *green*, *blue, alpha*)**

The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all):

## **HEX Value**

In CSS, a color can be specified using a hexadecimal value in the form:

**#rrggbb**

Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

To display black, set all values to 00, like this: #000000.

To display white, set all values to ff, like this: #ffffff.

Experiment by mixing the HEX values below:

For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).

To display black, set all values to 00, like this: #000000.

To display white, set all values to ff, like this: #ffffff.

Experiment by mixing the HEX values below:

## **HSL Value**

In CSS, a color can be specified using hue, saturation, and lightness (HSL) in the form:

**hsl(hue, saturation, lightness)**

Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.

Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.

Lightness is also a percentage. 0% is black, 50% is neither light or dark, 100% is white

Experiment by mixing the HSL values below:

## **CSS Margins**

The CSS margin properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 80px;  
}

p {  
  margin: 25px 50px 75px 100px;  
}

**The auto Value**

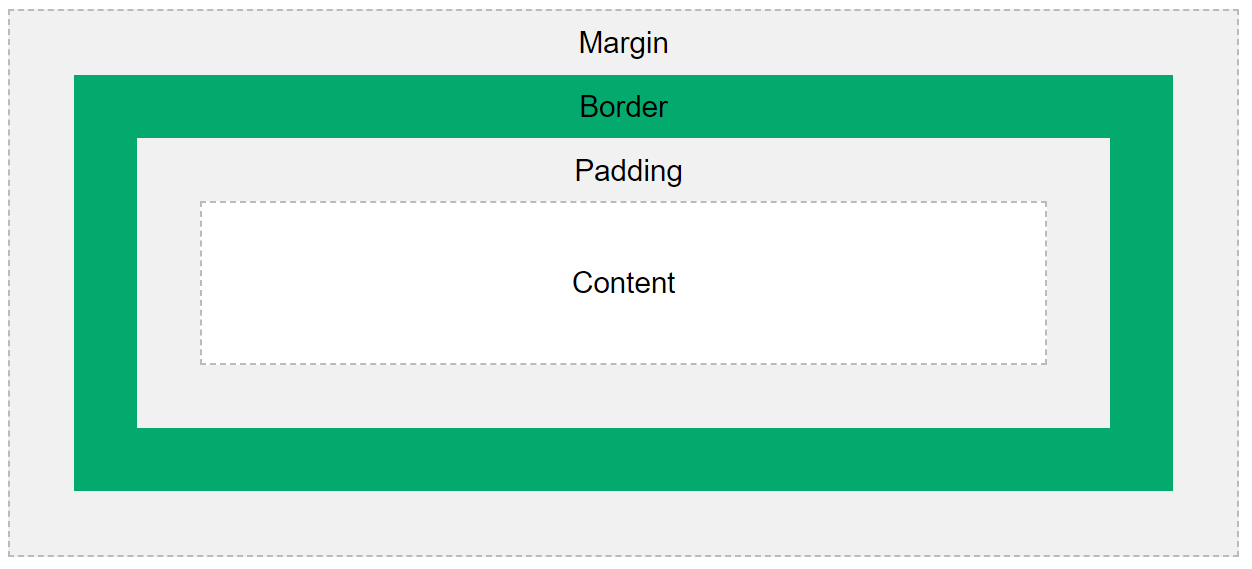
You can set the margin property to auto to horizontally center the element within its container.

div {  
  width: 300px;  
  margin: auto;  
  border: 1px solid red;  
}

# CSS Box Model

In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: content, padding, borders and margins. The image below illustrates the box model:



* **Content** - The content of the box, where text and images appear
* **Padding** - Clears an area around the content. The padding is transparent
* **Border** - A border that goes around the padding and content
* **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

div {  
  width: 300px;  
  border: 15px solid green;  
  padding: 50px;  
  margin: 20px;  
}

## **Width and Height of an Element**

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

**Important:** When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**. To calculate the total width and height of an element, you must also include the padding and borders.

### **Example**

This <div> element will have a total width of 350px and a total height of 80px:

div {  
  width: 320px;  
  height: 50px;  
  padding: 10px;  
  border: 5px solid gray;  
  margin: 0;  
}

Here is the calculation:

  320px (width of content area)  
+ 20px (left padding + right padding)  
+ 10px (left border + right border)  
**= 350px (total width)**

  50px (height of content area)  
+ 20px (top padding + bottom padding)  
+ 10px (top border + bottom border)  
**= 80px (total height)**  
  
  50px (height of content area)  
+ 20px (top padding + bottom padding)  
+ 10px (top border + bottom border)  
**= 80px (total height)**

The total width of an element should be calculated like this:

Total element width = width + left padding + right padding + left border + right border

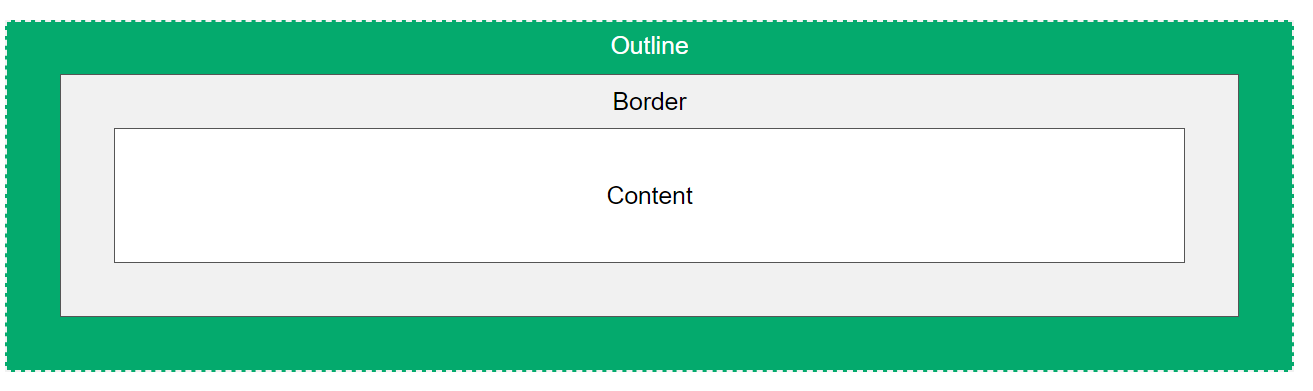
The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border

**Note:** The margin property also affects the total space that the box will take up on the page, but the margin is not included in the actual size of the box. The box's total width and height stops at the border.

# CSS Outline

An outline is a line drawn outside the element's border.



## **CSS Outline Style**

The outline-style property specifies the style of the outline, and can have one of the following values:

* dotted - Defines a dotted outline
* dashed - Defines a dashed outline
* solid - Defines a solid outline
* double - Defines a double outline
* groove - Defines a 3D grooved outline
* ridge - Defines a 3D ridged outline
* inset - Defines a 3D inset outline
* outset - Defines a 3D outset outline
* none - Defines no outline
* hidden - Defines a hidden outline

## **CSS Outline Width**

The outline-width property specifies the width of the outline, and can have one of the following values:

* thin (typically 1px)
* medium (typically 3px)
* thick (typically 5px)
* A specific size (in px, pt, cm, em, etc)

p.ex1 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: thin;  
}  
  
p.ex2 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: medium;  
}  
  
p.ex3 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: thick;  
}  
  
p.ex4 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: 4px;  
}

## **CSS Outline Color**

The outline-color property is used to set the color of the outline.

## **CSS Outline - Shorthand property**

The outline property is a shorthand property for setting the following individual outline properties:

* outline-width
* outline-style (required)
* outline-color

p.ex1 {outline: dashed;}  
p.ex2 {outline: dotted red;}  
p.ex3 {outline: 5px solid yellow;}  
p.ex4 {outline: thick ridge pink;}

## **CSS Outline Offset**

The outline-offset property adds space between an outline and the edge/border of an element. The space between an element and its outline is transparent.

The following example specifies an outline 15px outside the border edge:

**p {**

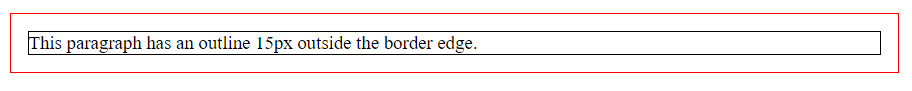
**margin: 30px;**

**border: 1px solid black;**

**outline: 1px solid red;**

**outline-offset: 15px;**

**}**



# CSS Text

## **Text Color**

The color property is used to set the color of the text. The color is specified by:

## **Text Color and Background Color**

In this example, we define both the background-color property and the color property

body {  
  background-color: lightgrey;  
  color: blue;  
}  
  
h1 {  
  background-color: black;  
  color: white;  
}  
  
div {  
  background-color: blue;  
  color: white;  
}

## **Text Alignment**

The text-align property is used to set the horizontal alignment of a text.

A text can be left or right aligned, centered, or justified.

When the text-align property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers):

## **Text Align Last**

The text-align-last property specifies how to align the last line of a text.

p.a {  
  text-align-last: right;  
}  
  
p.b {  
  text-align-last: center;  
}  
  
p.c {

  text-align-last: justify;

## **Text Direction**

The direction and unicode-bidi properties can be used to change the text direction of an element:

p.ex1 {  
  direction: rtl;  
  unicode-bidi: bidi-override;  
}

<p class="ex1">This is right-to-left text direction.</p>

In browser:



## **Vertical Alignment**

The vertical-align property sets the vertical alignment of an element.

img.a {  
  vertical-align: baseline;  
}  
  
img.b {  
  vertical-align: text-top;  
}  
  
img.c {  
  vertical-align: text-bottom;  
}  
  
img.d {  
  vertical-align: sub;  
}  
  
img.e {

  vertical-align: super;

**The CSS Text Alignment/Direction Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [text-decoration](https://www.w3schools.com/cssref/pr_text_text-decoration.asp) | Sets all the text-decoration properties in one declaration |
| [text-decoration-color](https://www.w3schools.com/cssref/css3_pr_text-decoration-color.asp) | Specifies the color of the text-decoration |
| [text-decoration-line](https://www.w3schools.com/cssref/css3_pr_text-decoration-line.asp) | Specifies the kind of text decoration to be used (underline, overline, etc.) |
| [text-decoration-style](https://www.w3schools.com/cssref/css3_pr_text-decoration-style.asp) | Specifies the style of the text decoration (solid, dotted, etc.) |
| [text-decoration-thickness](https://www.w3schools.com/cssref/pr_text_text-decoration-thickness.asp) | Specifies the thickness of the text decoration line |

## **All CSS text-decoration Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [direction](https://www.w3schools.com/cssref/pr_text_direction.asp) | Specifies the text direction/writing direction |
| [text-align](https://www.w3schools.com/cssref/pr_text_text-align.asp) | Specifies the horizontal alignment of text |
| [text-align-last](https://www.w3schools.com/cssref/css3_pr_text-align-last.asp) | Specifies how to align the last line of a text |
| [unicode-bidi](https://www.w3schools.com/cssref/pr_text_unicode-bidi.asp) | Used together with the [direction](https://www.w3schools.com/cssref/pr_text_direction.asp) property to set or return whether the text should be overridden to support multiple languages in the same document |
| [vertical-align](https://www.w3schools.com/cssref/pr_pos_vertical-align.asp) | Sets the vertical alignment of an element |

h1 {  
  text-decoration-line: overline;  
}  
  
h2 {  
  text-decoration-line: line-through;  
}  
  
h3 {  
  text-decoration-line: underline;  
}  
  
p {  
  text-decoration-line: overline underline;  
}

## **Specify a Color for the Decoration Line**

The text-decoration-color property is used to set the color of the decoration line.

## **Specify a Style for the Decoration Line**

The text-decoration-style property is used to set the style of the decoration line.

### **Example**

h1 {  
  text-decoration-line: underline;  
  text-decoration-style: solid;  
}  
  
h2 {  
  text-decoration-line: underline;  
  text-decoration-style: double;  
}  
  
h3 {  
  text-decoration-line: underline;  
  text-decoration-style: dotted;  
}  
  
p.ex1 {  
  text-decoration-line: underline;  
  text-decoration-style: dashed;  
}  
  
p.ex2 {  
  text-decoration-line: underline;  
  text-decoration-style: wavy;  
}  
  
p.ex3 {  
  text-decoration-line: underline;  
  text-decoration-color: red;  
  text-decoration-style: wavy;  
}

## **Specify the Thickness for the Decoration Line**

The text-decoration-thickness property is used to set the thickness of the decoration line.

## h2 {   text-decoration-line: underline;   text-decoration-thickness: 5px; }

## **The Shorthand Property**

The text-decoration property is a shorthand property for:

* text-decoration-line (required)
* text-decoration-color (optional)
* text-decoration-style (optional)
* text-decoration-thickness (optional)

h1 {  
  text-decoration: underline;  
}  
  
h2 {  
  text-decoration: underline red;  
}  
  
h3 {  
  text-decoration: underline red double;  
}  
  
p {  
  text-decoration: underline red double 5px;  
}

* text-indent text-indent: 50px;
* letter-spacing letter-spacing: 5px;
* line-height line-height: 1.8;
* word-spacing word-spacing: 10px;
* white-space white-space: nowrap;

## **Text Shadow**

The text-shadow property adds shadow to text.

## In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

h1 {  
  text-shadow: 2px 2px;  
}

h1 {  
  text-shadow: 2px 2px red;  
}

Add a blur effect (5px) to the shadow:

h1 {  
  text-shadow: 2px 2px 5px red;  
}

# CSS Links

In addition, links can be styled differently depending on what **state** they are in.

The four links states are:

* a:link - a normal, unvisited link
* a:visited - a link the user has visited
* a:hover - a link when the user mouses over it
* a:active - a link the moment it is clicked

/\* 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;  
}

When setting the style for several link states, there are some order rules:

* a:hover MUST come after a:link and a:visited
* a:active MUST come after a:hover

## **Text Decoration**

The text-decoration property is mostly used to remove underlines from links:

a:link {  
  text-decoration: none;  
}  
  
a:visited {  
  text-decoration: none;  
}  
  
a:hover {  
  text-decoration: underline;  
}  
  
a:active {  
  text-decoration: underline;  
}

## **Background Color**

The background-color property can be used to specify a background color for links:

### **Example**

a:link {  
  background-color: yellow;  
}  
  
a:visited {  
  background-color: cyan;  
}  
  
a:hover {  
  background-color: lightgreen;  
}  
  
a:active {  
  background-color: hotpink;  
}

## **Link Buttons**

This example demonstrates a more advanced example where we combine several CSS properties to display links as boxes/buttons:

a:link, a:visited {  
  background-color: #f44336;  
  color: white;  
  padding: 14px 25px;  
  text-align: center;  
  text-decoration: none;  
  display: inline-block;  
}  
  
a:hover, a:active {  
  background-color: red;  
}

# CSS Layout - The display Property

## **The display Property**

The display property is used to specify how an element is shown on a web page.

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.

The display property is used to change the default display behavior of HTML elements.

# CSS Lists

## **HTML Lists and CSS List Properties**

In HTML, there are two main types of lists:

* unordered lists (<ul>) - the list items are marked with bullets
* ordered lists (<ol>) - the list items are marked with numbers or letters

The CSS list properties allow you to:

* Set different list item markers for ordered lists
* Set different list item markers for unordered lists
* Set an image as the list item marker
* Add background colors to lists and list items

## **Different List Item Markers**

The list-style-type property specifies the type of list item marker.

The following example shows some of the available list item markers:

### **Example**

ul.a {  
  list-style-type: circle;  
}  
  
ul.b {  
  list-style-type: square;  
}  
  
ol.c {  
  list-style-type: upper-roman;  
}  
  
ol.d {  
  list-style-type: lower-alpha;  
}

## **An Image as The List Item Marker**

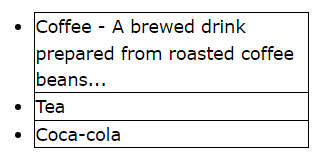
The list-style-image property specifies an image as the list item marker:

ul {  
  list-style-image: url('sqpurple.gif');  
}

## **Position The List Item Markers**

The list-style-position property specifies the position of the list-item markers (bullet points).

"list-style-position: outside;" means that the bullet points will be outside the list item. The start of each line of a



list item will be aligned vertically. This is default:

"list-style-position: inside;" means that the bullet points will be inside the list item. As it is part of the list item, it will be part of the text and push the text at the start:

* Coffee - A brewed drink prepared from roasted coffee beans...
* Tea
* Coca-cola

ul.a {  
  list-style-position: outside;  
}  
  
ul.b {  
  list-style-position: inside;  
}

# CSS Layout - The position Property

## **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.

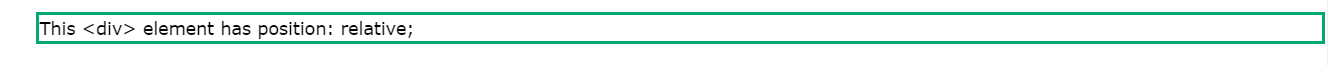
An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:Here is the CSS that is used:

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

## **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.



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

## **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;  
}

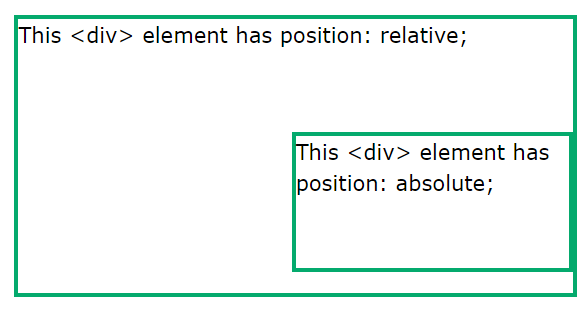
## **position: absolute;**

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:

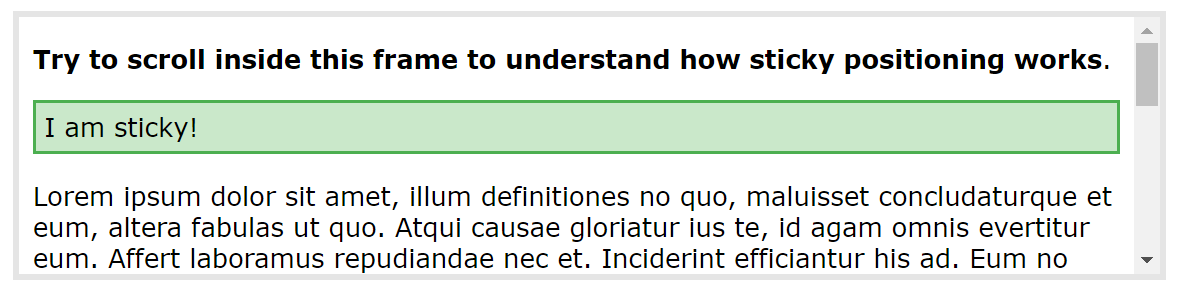


div.relative {  
  position: relative;  
  width: 400px;  
  height: 200px;  
  border: 3px solid #73AD21;  
}  
  
div.absolute {  
  position: absolute;  
  top: 80px;  
  right: 0;  
  width: 200px;  
  height: 100px;  
  border: 3px solid #73AD21;  
}

## **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.

# CSS Layout - 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:

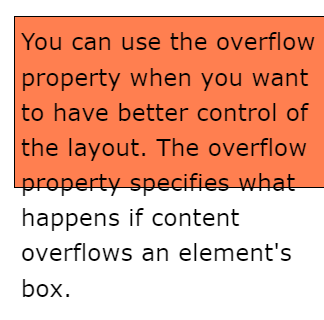
* visible - Default. The overflow is not clipped. The content renders outside the element's box
* hidden - The overflow is clipped, and the rest of the content will be invisible
* scroll - The overflow is clipped, and a scrollbar is added to see the rest of the content
* auto - Similar to scroll, but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

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

## **overflow: visible**

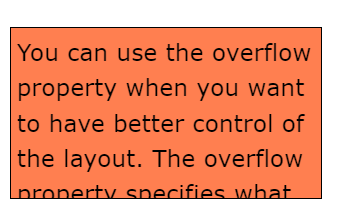
By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box:

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



## **overflow: hidden**

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



div {  
  overflow: hidden;  
}

## **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):

## **overflow: auto**

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

## **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.

div {  
  overflow-x: hidden; /\* Hide horizontal scrollbar \*/  
  overflow-y: scroll; /\* Add vertical scrollbar \*/  
}

## **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;  
}

## **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;  
}

<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>

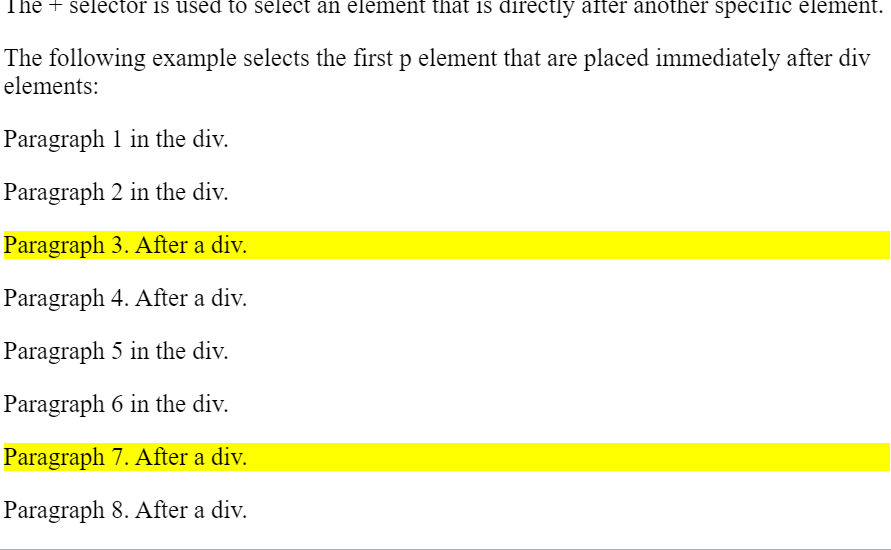
## **Adjacent Sibling Selector (+)**

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;  
}

<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>

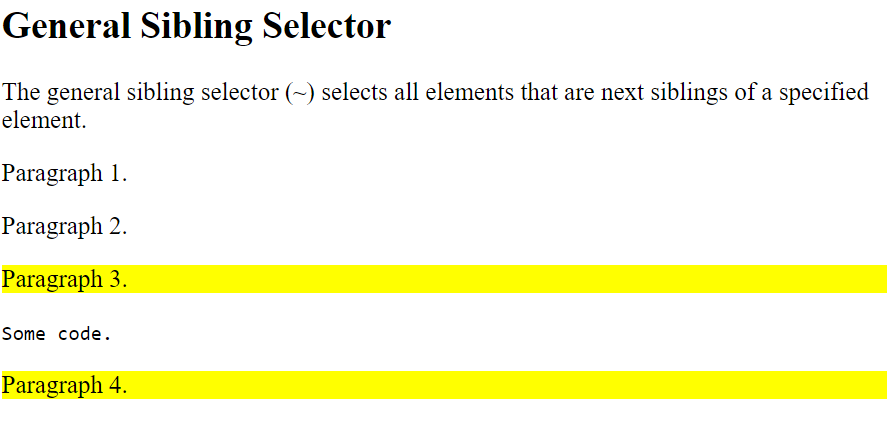
## **General Sibling Selector (~)**

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;  
}

<div>

<p>Paragraph 2.</p>

</div>

<p>Paragraph 3.</p>

<code>Some code.</code>

<p>Paragraph 4.</p>

# CSS Opacity / Transparency

The opacity property specifies the opacity/transparency of an element.

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

img {  
  opacity: 0.5;  
}

## **Transparent Hover Effect**

The opacity property is often used together with the :hover selector to change the opacity on mouse-over:

img {  
  opacity: 0.5;  
}  
  
img:hover {  
  opacity: 1.0;  
}

# CSS Backgrounds

In these chapters, you will learn about the following CSS background properties:

* background-color
* background-image
* background-repeat
* background-attachment
* background-position
* background

## **CSS background-color**

The background-color property specifies the background color of an element.

body {  
  background-color: lightblue;  
}

## **Opacity / Transparency**

The opacity property specifies the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent:

div {  
  background-color: green;  
  opacity: 0.3;  
}

**Note:** When using the opacity property to add transparency to the background of an element, all of its child elements inherit the same transparency. This can make the text inside a fully transparent element hard to read.

## **CSS background-image**

The background-image property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.

### **Example**

Set the background image for a page:

body {  
  background-image: url("paper.gif");  
}

## **CSS background-repeat**

By default, the background-image property repeats an image both horizontally and vertically.

Some images should be repeated only horizontally or vertically, or they will look strange, like this:

## If the image above is repeated only horizontally (background-repeat: repeat-x;), the background will look better:

body {  
  background-image: url("gradient\_bg.png");  
  background-repeat: repeat-x;  
}

**Tip:** To repeat an image vertically, set background-repeat: repeat-y;

## **CSS background-repeat: no-repeat**

Showing the background image only once is also specified by the background-repeat property:

body {  
  background-image: url("img\_tree.png");  
  background-repeat: no-repeat;  
}

In the example above, the background image is placed in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much.

## **CSS background-position**

The background-position property is used to specify the position of the background image.

Position the background image in the top-right corner:

body {  
  background-image: url("img\_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
}

## **CSS background-attachment**

The background-attachment property specifies whether the background image should scroll or be fixed (will not scroll with the rest of the page):

### **Example**

Specify that the background image should be fixed:

body {  
  background-image: url("img\_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
  background-attachment: fixed;  
}

### **Example**

Specify that the background image should scroll with the rest of the page:

body {  
  background-image: url("img\_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
  background-attachment: scroll;  
}

|  |  |
| --- | --- |
| **Value** | **Description** |
| scroll | The background image will scroll with the page. This is default |
| fixed | The background image will not scroll with the page |
| local | The background image will scroll with the element's contents |
| initial | Sets this property to its default value. [Read about initial](https://www.w3schools.com/cssref/css_initial.php) |
| inherit | Inherits this property from its parent element. [Read about inherit](https://www.w3schools.com/cssref/css_inherit.php) |

## **CSS background - Shorthand property**

To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

Instead of writing:

body {  
  background-color: #ffffff;  
  background-image: url("img\_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
}

shorthand:

body {  
  background: #ffffff url("img\_tree.png") no-repeat right top;  
}

## **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;  
}

## **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:

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

## **Hover on <div>**

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

div:hover {  
  background-color: blue;  
}

<p><a class="highlight" href="css\_syntax.asp">CSS Syntax</a></p>

## **Simple Tooltip Hover**

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

**Hover over me to show the <p> element.**

p {

display: none;

background-color: yellow;

padding: 20px;

}

div:hover p {

display: block;

}

<div>Hover over this div element to show the p element

<p>Tada! Here I am!</p>

## **What are Pseudo-Elements?**

A CSS pseudo-element is used to style specified parts of an element.

For example, it can be used to:

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

## **Syntax**

The syntax of pseudo-elements:

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

## **The ::first-line Pseudo-element**

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

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

p::first-line {

color: #ff0000;

font-variant: small-caps;

}

<p>You can use the ::first-line pseudo-element to add a special effect to the first line of a text. Some more text. And even more, and more, and more, and more, and more, and more, and more, and more, and more, and more, and more, and more.</p>

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

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

**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.

p::first-letter {

color: #ff0000;

font-size: xx-large;

}

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

## **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:

h1::before {

content: url(smiley.gif);

}

<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>

## **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);  
}

## **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;

}

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

<p>This is a paragraph.</p>

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

## **CSS Transitions**

CSS transitions allows you to change property values smoothly, over a given duration.

**Mouse over the element below to see a CSS transition effect:**

In this chapter you will learn about the following properties:

* transition
* transition-delay
* transition-duration
* transition-property
* transition-timing-function

## **How to Use CSS Transitions?**

To create a transition effect, you must specify two things:

* the CSS property you want to add an effect to
* the duration of the effect

**Note:** If the duration part is not specified, the transition will have no effect, because the default value is 0.

The following example shows a 100px \* 100px red <div> element. The <div> element has also specified a transition effect for the width property, with a duration of 2 seconds:

### **Example**

div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  transition: width 2s;  
}

The transition effect will start when the specified CSS property (width) changes value.

Now, let us specify a new value for the width property when a user mouses over the <div> element:

### **Example**

div:hover {  
  width: 300px;  
}

<h1>The transition Property</h1>

<p>Hover over the div element below, to see the transition effect:</p>

<div></div>

Notice that when the cursor mouses out of the element, it will gradually change back to its original style.

## **Change Several Property Values**

The following example adds a transition effect for both the width and height property, with a duration of 2 seconds for the width and 4 seconds for the height:

div {

width: 100px;

height: 100px;

background: red;

transition: width 2s, height 4s;

}

div:hover {

width: 300px;

height: 300px;

}

## **Specify the Speed Curve of the Transition**

The transition-timing-function property specifies the speed curve of the transition effect.

The transition-timing-function property can have the following values:

* ease - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
* linear - specifies a transition effect with the same speed from start to end
* ease-in - specifies a transition effect with a slow start
* ease-out - specifies a transition effect with a slow end
* ease-in-out - specifies a transition effect with a slow start and end
* cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier function

The following example shows some of the different speed curves that can be used:

### **Example**

#div1 {transition-timing-function: linear;}  
#div2 {transition-timing-function: ease;}  
#div3 {transition-timing-function: ease-in;}  
#div4 {transition-timing-function: ease-out;}  
#div5 {transition-timing-function: ease-in-out;}

## **Delay the Transition Effect**

The transition-delay property specifies a delay (in seconds) for the transition effect.

The following example has a 1 second delay before starting:

### **Example**

div {  
  transition-delay: 1s;  
}

## **Transition + Transformation**

The following example adds a transition effect to the transformation:

div {

width: 100px;

height: 100px;

background: red;

transition: width 2s, height 2s, transform 2s;

}

div:hover {

width: 300px;

height: 300px;

transform: rotate(180deg);

}

<h1>Transition + Transform</h1>

<p>Hover over the div element below:</p>

<div></div>

## **CSS Animations**

CSS allows animation of HTML elements without using JavaScript!

In this chapter you will learn about the following properties:

* @keyframes
* animation-name
* animation-duration
* animation-delay
* animation-iteration-count
* animation-direction
* animation-timing-function
* animation-fill-mode
* animation

## **What are CSS Animations?**

An animation lets an element gradually change from one style to another.

You can change as many CSS properties you want, as many times as you want.

To use CSS animation, you must first specify some keyframes for the animation.

Keyframes hold what styles the element will have at certain times.

## **The @keyframes Rule**

When you specify CSS styles inside the @keyframes rule, the animation will gradually change from the current style to the new style at certain times.

To get an animation to work, you must bind the animation to an element.

The following example binds the "example" animation to the <div> element. The animation will last for 4 seconds, and it will gradually change the background-color of the <div> element from "red" to "yellow":

/\* The animation code \*/  
@keyframes example {  
  from {background-color: red;}  
  to {background-color: yellow;}  
}  
  
/\* The element to apply the animation to \*/  
div {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
}

<h1>CSS Animation</h1>

<div></div>

<p><b>Note:</b> When an animation is finished, it goes back to its original style.</p>

**Note:** The animation-duration property defines how long an animation should take to complete. If the animation-duration property is not specified, no animation will occur, because the default value is 0s (0 seconds).

In the example above we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)).

It is also possible to use percent. By using percent, you can add as many style changes as you like.

The following example will change the background-color of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

/\* The animation code \*/  
@keyframes example {  
  0%   {background-color: red;}  
  25%  {background-color: yellow;}  
  50%  {background-color: blue;}  
  100% {background-color: green;}  
}  
  
/\* The element to apply the animation to \*/  
div {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
}

The following example will change both the background-color and the position of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:

@keyframes example {  
  0%   {background-color:red; left:0px; top:0px;}  
  25%  {background-color:yellow; left:200px; top:0px;}  
  50%  {background-color:blue; left:200px; top:200px;}  
  75%  {background-color:green; left:0px; top:200px;}  
  100% {background-color:red; left:0px; top:0px;}  
}  
  
/\* The element to apply the animation to \*/  
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
}

## **Delay an Animation**

The animation-delay property specifies a delay for the start of an animation.

The following example has a 2 seconds delay before starting the animation:

div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-delay: 2s;  
}

<h1>CSS Animation</h1>

<p>The animation-delay property specifies a delay for the start of an animation. The following example has a 2 seconds delay before starting the animation:</p>

<div></div>

Negative values are also allowed. If using negative values, the animation will start as if it had already been playing for N seconds.

In the following example, the animation will start as if it had already been playing for 2 seconds:

### **Example**

div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-delay: -2s;  
}

@keyframes example {

0% {background-color:red; left:0px; top:0px;}

25% {background-color:yellow; left:200px; top:0px;}

50% {background-color:blue; left:200px; top:200px;}

75% {background-color:green; left:0px; top:200px;}

100% {background-color:red; left:0px; top:0px;}

}

<h1>CSS Animation</h1>

<p>Using negative values in the animation-delay property: Here, the animation will start as if it had already been playing for 2 seconds:</p>

<div></div>

## **Set How Many Times an Animation Should Run**

The animation-iteration-count property specifies the number of times an animation should run.

The following example will run the animation 3 times before it stops:

div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-iteration-count: 3;  
}

The following example uses the value "infinite" to make the animation continue for ever:

 animation-iteration-count: infinite;

## **Run Animation in Reverse Direction or Alternate Cycles**

The animation-direction property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

* normal - The animation is played as normal (forwards). This is default
* reverse - The animation is played in reverse direction (backwards)
* alternate - The animation is played forwards first, then backwards
* alternate-reverse - The animation is played backwards first, then forwards

The following example will run the animation in reverse direction (backwards):

div {

width: 100px;

height: 100px;

background-color: red;

position: relative;

animation-name: example;

animation-duration: 4s;

animation-direction: reverse;

}

@keyframes example {

0% {background-color:red; left:0px; top:0px;}

25% {background-color:yellow; left:200px; top:0px;}

50% {background-color:blue; left:200px; top:200px;}

75% {background-color:green; left:0px; top:200px;}

100% {background-color:red; left:0px; top:0px;}

}

<h1>CSS Animation</h1>

<p>The animation-direction property specifies whether an animation should be played forwards, backwards or in alternate cycles. The following example will run the animation in reverse direction (backwards):</p>

<div></div>

The following example uses the value "alternate" to make the animation run forwards first, then backwards:

The following example uses the value "alternate-reverse" to make the animation run backwards first, then forwards:

## **Specify the Speed Curve of the Animation**

The animation-timing-function property specifies the speed curve of the animation.

The animation-timing-function property can have the following values:

* ease - Specifies an animation with a slow start, then fast, then end slowly (this is default)
* linear - Specifies an animation with the same speed from start to end
* ease-in - Specifies an animation with a slow start
* ease-out - Specifies an animation with a slow end
* ease-in-out - Specifies an animation with a slow start and end
* cubic-bezier(n,n,n,n) - Lets you define your own values in a cubic-bezier function

The following example shows some of the different speed curves that can be used:

### **Example**

#div1 {animation-timing-function: linear;}  
#div2 {animation-timing-function: ease;}  
#div3 {animation-timing-function: ease-in;}  
#div4 {animation-timing-function: ease-out;}  
#div5 {animation-timing-function: ease-in-out;}

## **Specify the fill-mode For an Animation**

CSS animations do not affect an element before the first keyframe is played or after the last keyframe is played. The animation-fill-mode property can override this behavior.

The animation-fill-mode property specifies a style for the target element when the animation is not playing (before it starts, after it ends, or both).

The animation-fill-mode property can have the following values:

* none - Default value. Animation will not apply any styles to the element before or after it is executing
* forwards - The element will retain the style values that is set by the last keyframe (depends on animation-direction and animation-iteration-count)
* backwards - The element will get the style values that is set by the first keyframe (depends on animation-direction), and retain this during the animation-delay period
* both - The animation will follow the rules for both forwards and backwards, extending the animation properties in both directions

The following example lets the <div> element retain the style values from the last keyframe when the animation ends:

div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  position: relative;  
  animation-name: example;  
  animation-duration: 3s;  
  animation-fill-mode: forwards;  
}

@keyframes example {

from {top: 0px;}

to {top: 200px; background-color: blue;}

}

<h1>CSS Animation</h1>

<p>Let the div element retain the style values set by the last keyframe when the animation ends:</p>

<div></div>

## **Animation Shorthand Property**

The example below uses six of the animation properties:

div {  
  animation-name: example;  
  animation-duration: 5s;  
  animation-timing-function: linear;  
  animation-delay: 2s;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
}

@keyframes example {

0% {background-color:red; left:0px; top:0px;}

25% {background-color:yellow; left:200px; top:0px;}

50% {background-color:blue; left:200px; top:200px;}

75% {background-color:green; left:0px; top:200px;}

100% {background-color:red; left:0px; top:0px;}

}

The same animation effect as above can be achieved by using the shorthand animation property:

div {

width: 100px;

height: 100px;

background-color: red;

position: relative;

animation: myfirst 5s linear 2s infinite alternate;

}