

# Cascading Style Sheets(css)

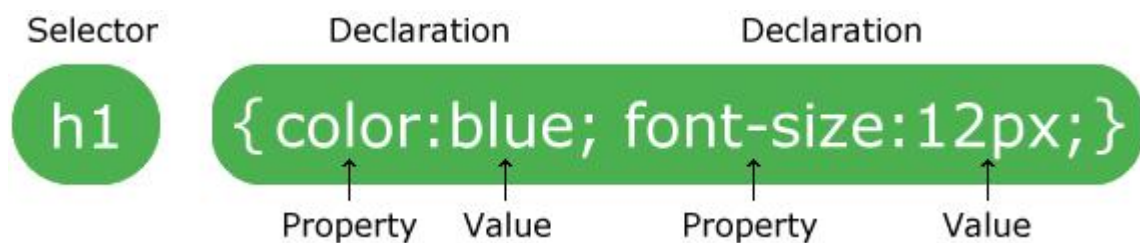
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**

## Why we use css?

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

### Syntax:

A CSS rule-set 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.

A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

Example:

```
P{
Color:red;
Text-align:center;
}
```

In the above example p is an element, color is property and red is the value

## 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)

Now we discuss about Simple Selector

## Simple selector

Simple select is again divided into five type based on different properties

- 1) Element selector
- 2) Class selector
- 3) Id selector
- 4) Universal selector
- 5) Group selector

### Element selector:

The element selector selects HTML elements (p,h1-h6,img,div..etc)based on the element name.

Example: In this <p> is an element , so css used to style that entire <p> element's in the page

```
P{
Color:red;
Text-align:center;
}
```

### Id selector:

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

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

Syntex 1:

```
#idname{
Color:red;
```

**Font-size:20px;**

**}**

In these, the style will apply for all the elements in the page only when we specify the id inside that particular element.

**<h1 id="idname"> CSS</h1>**(style applied)

**<p> Haii bro<p>** (style not applied)

Syntax2:

**Div#idname{**

**Color: blue;**

**Text-align:center;**

**}**

In these, the style apply only for specified <div> element only but not for other elements and we have to specify the <id> name inside the <div> so that the style will apply to it, other wise the style won't apply

**<div id="idname">**

**</div>**

## Class selector:

The class selector selects HTML elements with a specific class attribute(class="").

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

Syntax 1:

**.classname{**

**Font-style:italic;**

**}**

In these, the style will apply for all the elements in the page only when we specify the class inside that particular element.

**<div class="classname">**

Style applied

**</div>**

**<p> styled not applied</p>**

Syntax 2:

p.classname{

color:white;

Text-align:center;

}

**<p> style applied</p>**

In these, the style apply only for specified <p> element only but not for other elements and we have to specify the class name inside the <p> so that the style will apply to it, other wise the style won't apply

## Universal selector:

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

Syntax:

\*{

Font-style: italic;

Font-size:30px;

}

In these ,the style will be applies to all the elements in the page.

## Group selector:

If we want to apply same style for 2 to 3 elements then we use groupselector

Example:

P,h1,h5{

Color:red;

Font-style:italic; }

**In** the above example the style will apply for the elements which are specified in the selector

## Combinator selector

Combinator selector will explain about the relation between the two selector's

In css selectors, it contain more then one selector b/w the simple selector's and we can also include diff combinator's

In css, we have four diff combinator's

- 1) Decendant selector**
- 2) Child selector(>)**
- 3) Adjucent sibling selector(+)**
- 4) General adjucent selector(~)**

### **Decendant selector:**

The decendant selector matches all element that are decendant

Of specified element

Example:

```
Div p{
```

```
Color:red;
```

```
}
```

In the above example the style will apply to <p> elements which are inside the <div> element

```
<div>
```

```
<p> style is applied</P>
```

```
</div>
```

```
<p> style not applied</p>
```

### **Child selector:**

The child selector selects all the elements that childrens of that specified elements.

Example

```
Div p{
```

```
Color:red;
```

```
}
```

The style will apply to <p> elements which are children to the >div> elements like

```
<div>
```

```
<p> this is child<p>
```

```
<section><p> this is not children of div element</p></section>
```

```
</div>
```

### Adjacent sibling selector(+)

ASS selects all the elements that are adjacent sibling of a specified element. sibling elements must have same parent element (adjacent means immediate follow)

Example:

```
Div+p{
```

```
Font-size:30px;
```

```
}
```

```
<div>
```

```
<p> No effects</p>
```

```
</div>
```

```
<p> style effected</p>
```

The style will apply to elements which come immediately to <div> element.

## General sibling selector(~):

GSS selects all elements which are siblings of specified element

Example:

```
Div~p{
```

**Color:red;**

**}**

**<div>**

**<p>no effect</p>**

**<p>no effect</p>**

**</div>**

**<p>no effect</p>**

**<code> some other elements</code>**

**<p>effected</p>**

## **How to add Css to Html**

Css will be applied to html based on the situation. there are three ways to apply style to html page they are

- 1) Inline(attribute)
- 2) Internal(selector)
- 3) External(Style sheet)

### **Inline**

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.

Syntax:

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

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

### **Internal(selector)**

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.

```
h1 {  
    color: maroon;  
    margin-left: 40px;  
}  
  
</style>  
  
</head>  
  
<body>  
  
<h1>This is a heading</h1>  
  
<p>This is a paragraph.</p>
```

## External(Style sheet)

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.

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

```
<link rel="stylesheet" type="text/css" href="mystyle.css">  
  
</head>  
  
<body>  
  
<h1>This is a heading</h1>  
  
<p>This is a paragraph.</p>  
  
</body>
```

**Note:** Do not add a space between the property value and the unit (such as `margin-left: 20 px;`). The correct way is: `margin-left: 20px;`

**Imp:** If we apply Inline, internal, external to same element then page take inline style as first priority and second priority to internal and then to external style process.



# Comments

To comments an Css style code then use `/*-----*/`

## Colors

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

**RGB-** these color name is based on the red green blue color density values ,if we want to use yellow then rgb value is rgb(241,196,14)

**RGBA-**If we add opacity to the rgb then it is rgba value

## Backgrounds

The CSS background properties are used to define the background effects for elements.

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

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

### background-color

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

Example:

```
<head>
<style>
body {
  background-color: lightblue;
}
</style>
</head>
```

With CSS, a color is most often specified by:

- a valid color name - like "red"

- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

**Note:** We can also apply the background color for other elements also like div,p,h1-h6 etc

Example:

```
Div{  
  
Background-color:white;  
  
}
```

## 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:



Syntax:

```
div.first {  
  
opacity: 0.1;  
  
}
```

**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.

## 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.

Syntax:

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

## 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

Example 1:

```
body {  
  background-image: url("gradient_bg.png");  
  background-repeat: repeat-x;  
}
```

Here, a background image is repeated only horizontally!

Example 2:

```
body {  
  background-image: url("gradient_bg.png");  
  background-repeat: repeat-y;  
}
```

Here, a background image is repeated only vertically.

Example 3:

```
body {  
  background-image: url("gradient_bg.png");  
  background-repeat: no-repeat;  
}
```

Here, background image won't repeated .

# background-position

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

Syntax:

```
body {  
  
    background-image: url("img_tree.png");  
  
    background-repeat: no-repeat;  
  
    background-position: right top;  
  
    margin-right: 200px;  
  
}
```

Position the background image in the top-right corner:

# 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 1:

```
body {  
  
    background-image: url("img_tree.png");  
  
    background-repeat: no-repeat;  
  
    background-position: right top;  
  
    margin-right: 200px;  
  
    background-attachment: fixed;  
  
}
```

Specify that the background image should be fixed:

Example 2:

```
body {  
  
    background-image: url("img_tree.png");
```

```
background-repeat: no-repeat;

background-position: right top;

margin-right: 200px;

background-attachment: scroll;
}
```

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

## 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;

}
```

We can also write like this shortcut:

```
body {

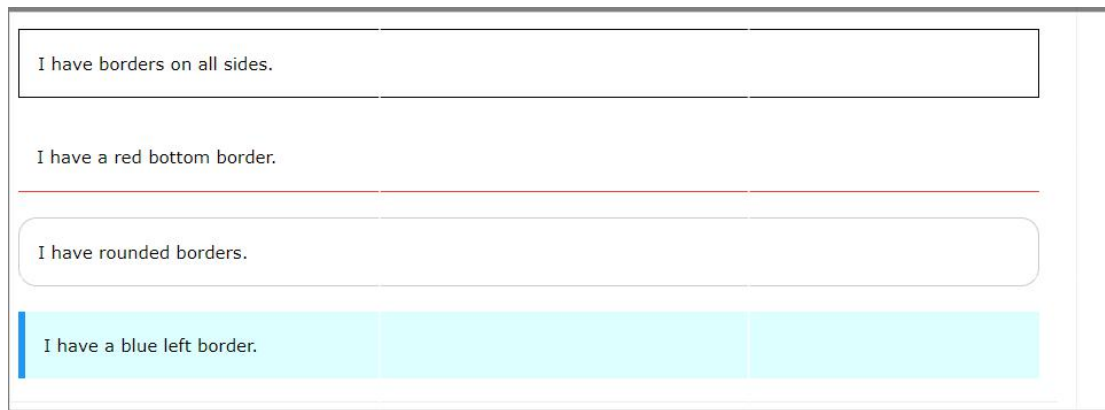
background: #ffffff url("img_tree.png") no-repeat right top;

margin-right: 200px;

}
```

## Borders

The CSS **border** properties allow you to specify the style, width, and color of an element's border.



## Border Style

The `border-style` property specifies what kind of border to display.

The following values are allowed:

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color value
- `inset` - Defines a 3D inset border. The effect depends on the border-color value
- `outset` - Defines a 3D outset border. The effect depends on the border-color value
- `none` - Defines no border
- `hidden` - Defines a hidden border

The `border-style` property can have from one to four values (for the top border, right border, bottom border, and the left border).

A dotted border.		
A dashed border.		
A solid border.		
A double border.		
A groove border. The effect depends on the border-color value.		
A ridge border. The effect depends on the border-color value.		
An inset border. The effect depends on the border-color value.		
An outset border. The effect depends on the border-color value.		
No border.		
A hidden border.		
A mixed border.		

### Syntax:

```
p.dotted {border-style: dotted;}
```

```
p.dashed {border-style: dashed;}
```

## Border Width

The `border-width` property specifies the width of the four borders.

The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick:

### **Syntax:**

```
p.one {
  border-style: solid;
  border-width: 5px;
}
```

## Specific Side Widths

The `border-width` property can have from one to four values (for the top border, right border, bottom border, and the left border)

## Syntax:

```
p.one {  
  
    border-style: solid;  
  
    border-width: 5px 20px; /* 5px top and bottom, 20px on the sides */  
}
```

```
p.two {  
  
    border-style: solid;  
  
    border-width: 20px 5px; /* 20px top and bottom, 5px on the sides */  
}
```

## Border Color

The `border-color` property is used to set the color of the four borders.

The color can be set by:

- name - specify a color name, like "red"
- HEX - specify a HEX value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- HSL - specify a HSL value, like "hsl(0, 100%, 50%)"
- transparent

**Note:** If `border-color` is not set, it inherits the color of the element.

## Syntax:

```
p.one {  
  
    border-style: solid;  
  
    border-color: red;  
}
```



# Specific Side Colors

The `border-color` property can have from one to four values (for the top border, right border, bottom border, and the left border).

## Syntax:

```
p.one {  
    border-style: solid;  
    border-color: red green blue yellow; /* red top, green right, blue  
    bottom and yellow left */  
}
```

# Border - Individual Sides

From the examples on the previous pages, you have seen that it is possible to specify a different border for each side.

In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left):

```
p {  
    border-top-style: dotted;  
    border-right-style: solid;  
    border-bottom-style: dotted;  
    border-left-style: solid;  
}
```



# Rounded Borders

The `border-radius` property is used to add rounded borders to an element:

Normal border

Round border

Rounder border

Roudest border

### Syntax:

```
p.normal {
```

```
  border: 2px solid red;
```

```
}
```

```
p.round1 {
```

```
  border: 2px solid red;
```

```
  border-radius: 5px;
```

```
}
```

```
p.round2 {
```

```
  border: 2px solid red;
```

```
  border-radius: 8px;
```

```
}
```

```
p.round3 {
```

```
  border: 2px solid red;
```

```
  border-radius: 12px;
```

```
}
```

# 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).

## Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

All the margin properties can have the following values:

- `auto` - the browser calculates the margin
- `length` - specifies a margin in px, pt, cm, etc.
- `%` - specifies a margin in % of the width of the containing element
- `inherit` - specifies that the margin should be inherited from the parent element

### Syntax:

```
div {  
  
    border: 1px solid black;  
  
    margin-top: 100px;  
  
    margin-bottom: 100px;  
  
    margin-right: 150px;  
  
    margin-left: 80px;  
  
    background-color: lightblue;  
  
}
```

### SHORTCUT:

```
div {  
  
    border: 1px solid black;  
  
    margin: 25px 50px 75px 100px;  
  
    background-color: lightblue;  
  
}
```

## Syntax 2:

```
div {
```

```
border: 1px solid black;
```

```
margin: 25px 50px 75px;
```

```
background-color: lightblue;
```

```
}
```

- **margin: 25px 50px 75px;**
  - top margin is 25px
  - right and left margins are 50px
  - bottom margin is 75px

---

## Syntax 3:

```
div {
```

```
border: 1px solid black;
```

```
margin: 25px 50px;
```

```
background-color: lightblue;
```

```
}
```

**margin: 25px 50px;**

- top and bottom margins are 25px
- right and left margins are 50px

## Syntax 4:

```
div {
```

```
border: 1px solid black;
```

```
margin: 25px;
```

```
background-color: lightblue;
```

```
}
```

**margin: 25px;**

- all four margins are 25px

## The auto Value

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

The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

### Syntax :

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

## The inherit Value

This example lets the left margin of the `<p class="ex1">` element be inherited from the parent element (`<div>`):

```
div {  
  border: 1px solid red;  
  margin-left: 100px;  
}
```

```
p.ex1 {  
  margin-left: inherit;  
}
```

## Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on left and right margins! Only top and bottom margins!

```
h1 {  
    margin: 0 0 50px 0;  
}
```

```
h2 {  
    margin: 20px 0 0 0;  
}
```

# Padding

The CSS `padding` properties are used to generate space around an element's content, inside of any defined borders.

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

## Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

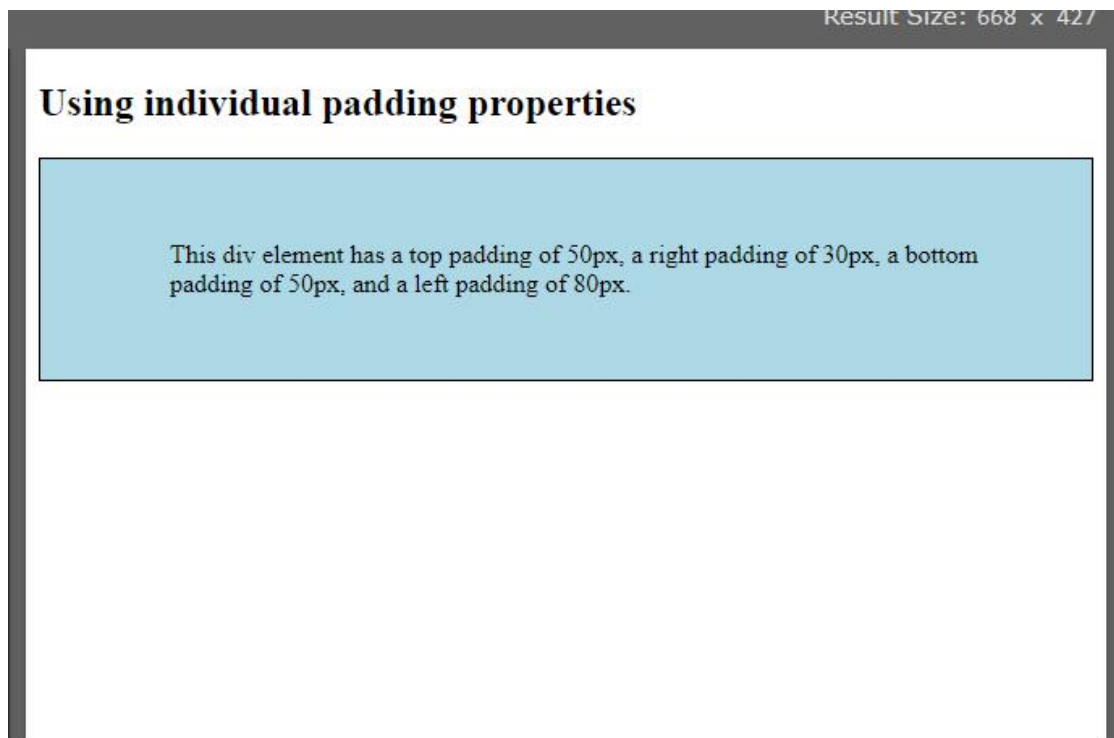
All the padding properties can have the following values:

- length - specifies a padding in px, pt, cm, etc.
- % - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

**Note:** Negative values are not allowed.

```
div {  
    border: 1px solid black;  
    background-color: lightblue;
```

```
padding-top: 50px;  
padding-right: 30px;  
padding-bottom: 50px;  
padding-left: 80px;  
}
```



### SHORTCUT:

```
div {  
border: 1px solid black;  
padding: 25px 50px 75px 100px;  
background-color: lightblue;  
}
```

Note: We can give any values upto 4 for padding which is same as margin

## Padding and Element Width

The CSS `width` property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element ([the box model](#)).

So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

## Syntax:

```
div.ex2 {  
  
    width: 300px;  
  
    padding: 25px;  
  
    background-color: lightblue;  
  
}
```

## Setting height and width

The `height` and `width` properties are used to set the height and width of an element.

The height and width properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element.

---

## CSS height and width Values

The `height` and `width` properties may have the following values:

- `auto` - This is default. The browser calculates the height and width
- `length` - Defines the height/width in px, cm etc.
- `%` - Defines the height/width in percent of the containing block
- `initial` - Sets the height/width to its default value
- `inherit` - The height/width will be inherited from its parent value

```
div {  
  
    height: 200px;  
  
    width: 50%;  
  
}
```



```
background-color: powderblue;  
}
```

This element has a height of 200 pixels and a width of 50%

## Setting max-width

The `max-width` property is used to set the maximum width of an element.

The `max-width` can be specified in length values, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).

The problem with the `<div>` above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.

Using `max-width` instead, in this situation, will improve the browser's handling of small windows.

**Note:** The value of the `max-width` property overrides `width`.

```
div {  
    max-width: 500px;  
    height: 100px;  
    background-color: powderblue;  
}
```

# Box Model



Explanation of the different parts:

- **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

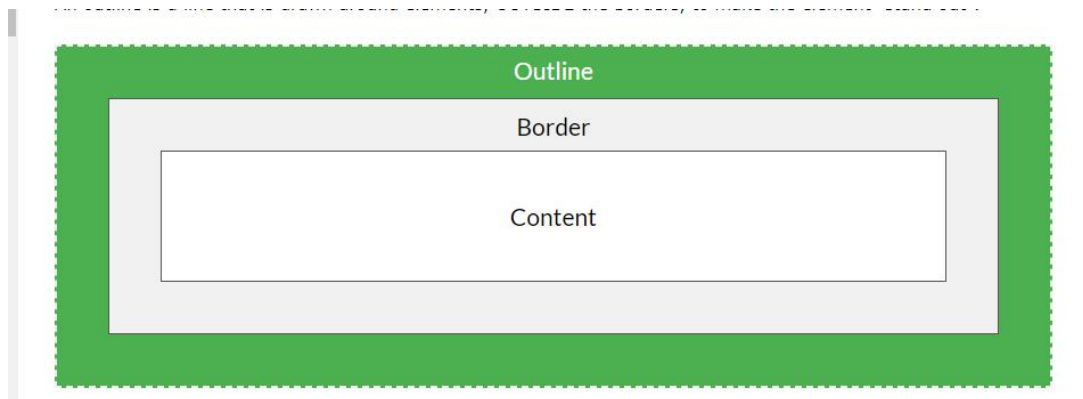
The box model allows us to add a border around elements, and to define space between elements.

## Syntax:

```
div {  
  background-color: lightgrey;  
  width: 300px;  
  border: 15px solid green;  
  padding: 50px;  
  margin: 20px;  
}
```

# Outline

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".



CSS has the following outline properties:

- `outline-style`
- `outline-color`
- `outline-width`
- `outline-offset`
- `outline`

**Note:** Outline differs from [borders](#)! Unlike border, the outline is drawn outside the element's border, and may overlap other content. Also, the outline is NOT a part of the element's dimensions; the element's total width and height is not affected by the width of the outline.

---

## 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

## Syntax:

```
p {outline-color:red;}
```

# 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)

## Syntax:

```
p.ex1 {  
  
  border: 1px solid black;  
  
  outline-style: solid;  
  
  outline-color: red;  
  
  outline-width: thin;  
  
}
```

# Outline Color

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

The color can be set by:

- name - specify a color name, like "red"
- HEX - specify a hex value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- HSL - specify a HSL value, like "hsl(0, 100%, 50%)"
- invert - performs a color inversion (which ensures that the outline is visible, regardless of color background)

## Syntax:

```
p.ex1 {  
  
  border: 2px solid black;
```

```
outline-style: solid;
```

```
outline-color: red;
```

```
}
```

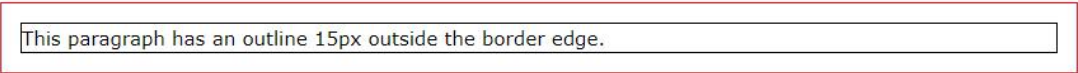
## SHORTCUT:

```
p.ex3 {outline: 5px dotted yellow;}
```

# 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:



This paragraph has an outline 15px outside the border edge.

```
p {
```

```
margin: 30px;
```

```
border: 1px solid black;
```

```
outline: 1px solid red;
```

```
outline-offset: 15px;
```

```
}
```

# Text

## Text Color

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

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

**Note:** For W3C compliant CSS: If you define the `color` property, you must also define the `background-color`. Because color of text must be different from background color.

```
body {  
  
  background-color: lightgrey;  
  
  color: blue;  
  
}
```

## 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.

```
h1 {  
  
  text-align: center;  
  
}
```

```
h2 {  
  
  text-align: left;  
  
}
```

```
h3 {  
  
  text-align: right;  
  
}
```

Note: There is one more value for text-align that is justified which make's text alignment like newspaper text alignment

## Text Decoration

The `text-decoration` property is used to set or remove decorations from text.

The value `text-decoration: none;` is often used to remove underlines from links:

```
a {
```

```
text-decoration: none;
```

```
}
```

The other `text-decoration` values are used to decorate text are Underline, Linethrough, Overline

**Note:** It is not recommended to underline text that is not a link, as this often confuses the reader.

## Text Transformation

The `text-transform` property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word

```
p.uppercase {
```

```
text-transform: uppercase;
```

```
}
```

```
p.lowercase {
```

```
text-transform: lowercase;
```

```
}
```

```
p.capitalize {
```

```
text-transform: capitalize;
```

```
}
```

Capitalize: Which make the first letter capital in ever

## Text Indentation

The `text-indent` property is used to specify the indentation of the first line of a text:

Indent means space at the starting of the paragraph

```
p {  
  
    text-indent: 50px;  
  
}
```

## Letter Spacing

The `letter-spacing` property is used to specify the space between the characters in a text.

```
h1 {  
  
    letter-spacing: 3px;  
  
}
```

## Line Height

The `line-height` property is used to specify the space between lines:

```
p.big {  
  
    line-height: 1.8;  
  
}
```

## Word Spacing

The `word-spacing` property is used to specify the space between the words in a text.

```
h2 {  
  
    word-spacing: -5px;  
  
}
```

## 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):



## Text shadow effect!

### Example

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

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

[Try it Yourself »](#)

Next, add a color (red) to the shadow:

## Text shadow effect!

### Example

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

[Try it Yourself »](#)

Then, add a blur effect (5px) to the shadow:

## Text shadow effect!

### Example

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

[Try it Yourself »](#)

# Fonts

## Font Family

The font family of a text is set with the `font-family` property.

The **font-family** property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font, and so on.

Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

**Note:** If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".

More than one font family is specified in a comma-separated list:

```
.serif {  
    font-family: "Times New Roman", Times, serif;  
}
```

```
.sansserif {  
    font-family: Arial, Helvetica, sans-serif;  
}
```

```
.monospace {  
    font-family: "Lucida Console", Courier, monospace;  
}
```

## Font Style

The **font-style** property is mostly used to specify italic text.

This property has three values:

- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

```
p.normal {  
    font-style: normal;
```

```
}
```

```
p.italic {
```

```
font-style: italic;
```

```
}
```

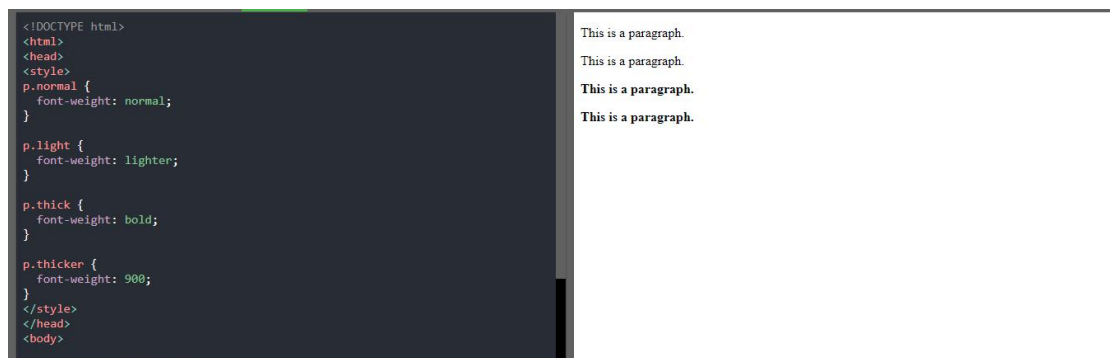
```
p.oblique {
```

```
font-style: oblique;
```

```
}
```

## Font Weight

The `font-weight` property specifies the weight of a font:



## Font Size

The `font-size` property sets the size of the text.

Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like `<h1>` - `<h6>` for headings and `<p>` for paragraphs.

The font-size value can be an absolute, or relative size.

Absolute size:

- Sets the text to a specified size

- Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
- Absolute size is useful when the physical size of the output is known

Relative size:

- Sets the size relative to surrounding elements
- Allows a user to change the text size in browsers

**Note:** If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

---

## Set Font Size With Pixels

Setting the text size with pixels gives you full control over the text size:

```
p {  
    font-size: 14px;  
}
```

Values for these property can be px or em or % as the developer wish

\*\*\*1em is equal to the current font size. The default text size in browsers is 16px. So, the default size of 1em is 16px.\*\*\*

## Responsive Font Size

The text size can be set with a **vw** unit, which means the "viewport width".

Note: Viewport is the browser window size. 1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

The **font** property is a shorthand property for:

- font-style
- font-variant
- font-weight
- font-size/line-height
- font-family

```
p.b {  
    font: italic bold 12px/30px Georgia, serif; }
```

# Links

With CSS, links can be styled in different ways.

[Text Link](#)

[Text Link](#)

[Link Button](#)

[Link Button](#)

## Styling Links

Links can be styled with any CSS property (e.g. `color`, `font-family`, `background`, etc.).

### Example

```
a {  
  color: hotpink;  
}
```

[Try it Yourself »](#)

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

---

Note: When you are using `<a href>` we must use text-decoration

## Background Color

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

---

```
a:link {
```

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

```
}
```

```
a:visited {
```

```
    background-color: cyan;
```

```
}
```

```
a:hover {
```

```
    background-color: lightgreen;
```

```
}
```

```
a:active {
```

```
    background-color: hotpink;}
```

# LinkButtons

```
<doctype html>
<html>
<head>
<style>
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;
}
```

## Link Button

A link styled as a button:

This is a link

# Lists

## Unordered Lists:

- Coffee
- Tea
- Coca Cola
  
- Coffee
- Tea
- Coca Cola

## Ordered Lists:

1. Coffee
2. Tea
3. Coca Cola
  
- I. Coffee
- II. Tea
- III. Coca Cola

# 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

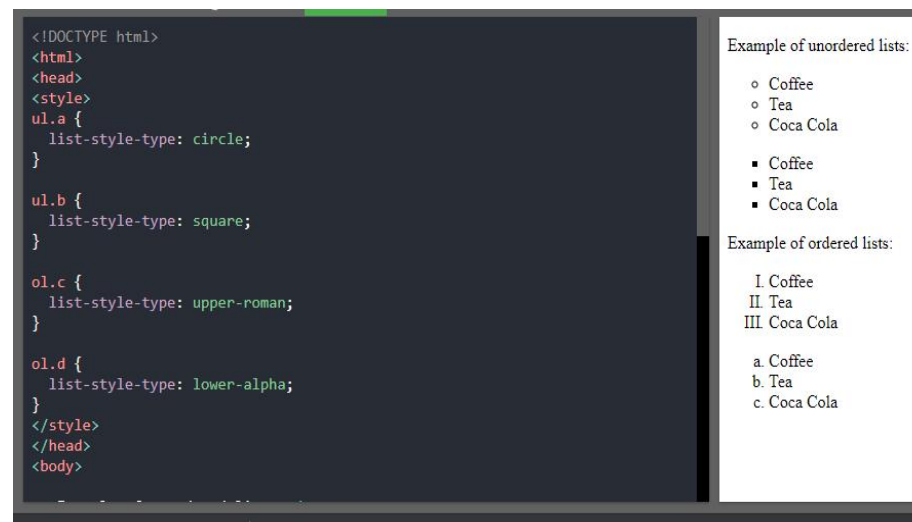
The CSS list properties allow you to:

- 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.



## 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



```
ul.a {
```

```
list-style-position: outside;
```

```
}
```

```
ul.b {
```

```
list-style-position: inside;
```

```
}
```

## 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:

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

"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   |

## Remove Default Settings

The `list-style-type:none` property can also be used to remove the markers/bullets. Note that the list also has default margin and padding. To remove this, add `margin:0` and `padding:0` to `<ul>` or `<ol>`:

```
ul.demo {
```

```
list-style-type: none;
```

```
margin: 0;
```

```
padding: 0;
```

```
}
```

# Tables

## Collapse Table Borders

The `border-collapse` property sets whether the table borders should be collapsed into a single border:

Firstname	Lastname
Peter	Griffin
Lois	Griffin

### Example

```
table {  
  border-collapse: collapse;  
}  
  
table, th, td {  
  border: 1px solid black;  
}
```

## Table Borders

To specify table borders in CSS, use the `border` property.

The example below specifies a black border for `<table>`, `<th>`, and `<td>` elements:

Firstname	Lastname
Peter	Griffin
Lois	Griffin

### Example

```
table, th, td {  
  border: 1px solid black;  
}
```

## Horizontal Alignment

The `text-align` property sets the horizontal alignment (like left, right, or center) of the content in `<th>` or `<td>`.

By default, the content of `<th>` elements are center-aligned and the content of `<td>` elements are left-aligned.

The following example left-aligns the text in `<th>` elements:

Firstname	Lastname	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

### Example

```
th {  
  text-align: left;  
}
```

The `vertical-align` property sets the vertical alignment (like top, bottom, or middle) of the content in `<th>` or `<td>`.

By default, the vertical alignment of the content in a table is middle (for both `<th>` and `<td>` elements).

The following example sets the vertical text alignment to bottom for `<td>` elements:

Firstname	Lastname	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

### Example

```
td {  
  height: 50px;  
  vertical-align: bottom;  
}
```

If you only want a border around the table, only specify the `border` property for `<table>`:

Firstname	Lastname
Peter	Griffin
Lois	Griffin

### Example

```
table {  
  border: 1px solid black;  
}
```

[Try it Yourself »](#)

## Table Width and Height

Width and height of a table are defined by the `width` and `height` properties.

The example below sets the width of the table to 100%, and the height of the `<th>` elements to 50px:

Firstname	Lastname	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

### Example

```
table {  
  width: 100%;  
}  
  
th {  
  height: 50px;  
}
```

## Table Padding

To control the space between the border and the content in a table, use the `padding` property on `<td>` and `<th>` elements:

Firstname	Lastname	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

### Example

```
th, td {  
  padding: 15px;  
  text-align: left;  
}
```

## Horizontal Dividers

First Name	Last Name	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

Add the `border-bottom` property to `<th>` and `<td>` for horizontal dividers:

### Example

```
th, td {  
  border-bottom: 1px solid #ddd;  
}
```

## Table Color

The example below specifies the background color and text color of `<th>` elements:

First Name	Last Name	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

### Example

```
th {  
  background-color: #4CAF50;  
  color: white;  
}
```

## Striped Tables

First Name	Last Name	Savings
Peter	Griffin	\$100
Lois	Griffin	\$150
Joe	Swanson	\$300

For zebra-striped tables, use the `nth-child()` selector and add a `background-color` to all even (or odd) table rows:

### Example

```
tr:nth-child(even) {background-color: #f2f2f2;}
```

## Responsive Table

A responsive table will display a horizontal scroll bar if the screen is too small to display the full content:

First Name	Last Name	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points	Poi
Jill	Smith	50	50	50	50	50	50	50	50	50	50	50
Eve	Jackson	94	94	94	94	94	94	94	94	94	94	94
Adam	Johnson	67	67	67	67	67	67	67	67	67	67	67

Add a container element (like `<div>`) with `overflow-x:auto` around the `<table>` element to make it responsive:

### Example

```
<div style="overflow-x:auto;">
  <table>
    ... table content ...
  </table>
</div>
```

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

# 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 <span> element inside` a paragraph.

Examples of inline elements:

- `<span>`
- `<a>`
- `<img>`

---

## Display: none;

`display: none;` is commonly used with JavaScript to hide and show elements without deleting and recreating them. Take a look at our last example on this page if you want to know how this can be achieved.

The `<script>` element uses `display: none;` as default.

---

## 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 `<li>` elements for horizontal menus:

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

Which makes the list to display in horizontal line not in vertical order

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

Which hides the h1 from the user, it will remove its space

`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:

# width and max-width

## Using width, max-width and margin: auto;

As mentioned in the previous chapter; a block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Setting the `width` of a block-level element will prevent it from stretching out to the edges of its container. Then, you can set the margins to auto, to horizontally center the element within its container. The element will take up the specified width, and the remaining space will be split equally between the two margins:

This `<div>` element has a width of 500px, and margin set to auto.

**Note:** The problem with the `<div>` above occurs when the browser window is smaller than the width of the element. The browser then adds a horizontal scrollbar to the page.

Using `max-width` instead, in this situation, will improve the browser's handling of small windows. This is important when making a site usable on small devices:

This `<div>` element has a max-width of 500px, and margin set to auto.

```
div.ex1 {  
  width:500px;  
  margin: auto;
```



```
border: 3px solid #73AD21;  
}
```

```
div.ex2 {  
  max-width:500px;  
  margin: auto;  
  border: 3px solid #73AD21;  
}
```

**Tip:** Resize the browser window to less than 500px wide, to see the difference between the two divs!

# 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:

This <div> element has position: static;

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.

This <div> element has position: relative;

```
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.

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

## position: absolute;

```
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;  
}
```

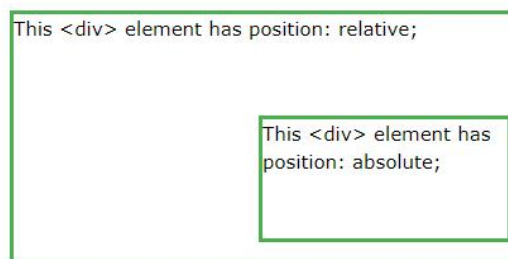
## position: absolute;

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:** A "positioned" element is one whose position is anything except `static`.

Here is a simple example:



## 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`).

```
div.sticky {  
  
    position: -webkit-sticky;  
  
    position: sticky;  
  
    top: 0;  
  
    padding: 5px;  
  
    background-color: #cae8ca;  
  
    border: 2px solid #4CAF50;  
  
}
```

## Overlapping Elements

When elements are positioned, they can overlap other elements.

The `z-index` property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order:



# This is a heading

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

### Example

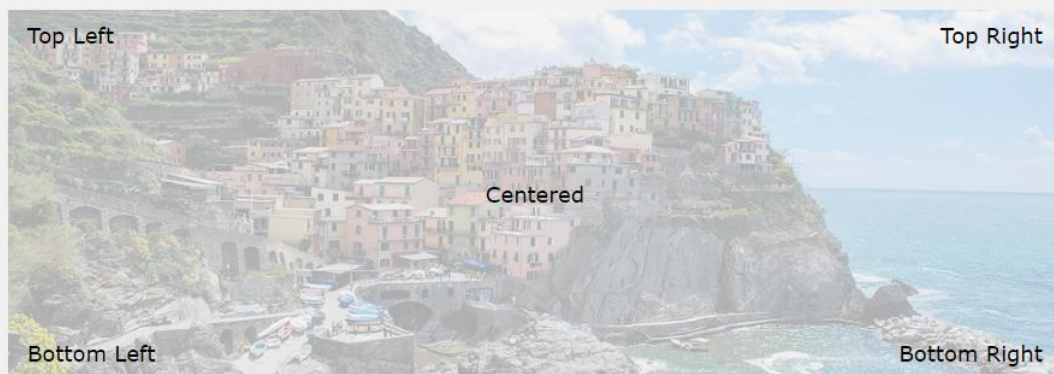
```
img {  
  position: absolute;  
  left: 0px;  
  top: 0px;  
  z-index: -1;  
}
```

This <div> element

## Positioning Text In an Image

How to position text over an image:

### Example



### Top Left:

```
.container {  
  position: relative;  
}
```

```
.topleft {  
  position: absolute;  
  top: 8px;  
  left: 16px;  
  font-size: 18px;  
}
```

```
img {  
  width: 100%;  
  height: auto;  
  opacity: 0.3;  
}
```

Top Right:

```
.container {  
  position: relative;  
}
```

```
.topright {  
  position: absolute;  
  top: 8px;  
  right: 16px;  
  font-size: 18px;  
}
```

```
img {  
  width: 100%;  
  height: auto;  
  opacity: 0.3;  
}
```

Center:

```
.container {  
  position: relative;  
}
```

```
.center {  
  position: absolute;  
  left: 0;  
  top: 50%;  
  width: 100%;  
  text-align: center;  
  font-size: 18px;  
}
```

```
img {  
  width: 100%;  
  height: auto;  
  opacity: 0.3;  
}
```

Bottom Left:

```
.container {  
  position: relative;  
}
```

```
.bottomleft {  
  position: absolute;  
  bottom: 8px;
```

```
left: 16px;  
font-size: 18px;  
}
```

```
img {  
width: 100%;  
height: auto;  
opacity: 0.3;  
}
```

## Bottom Right:

```
.container {  
position: relative;  
}
```

```
.bottomright {  
position: absolute;  
bottom: 8px;  
right: 16px;  
font-size: 18px;  
}
```

```
img {  
width: 100%;  
height: auto;  
opacity: 0.3;  
}
```

# Layout - Overflow

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

## 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.

--> In these the overflow content will be visible like, in the above case 4 and 5 lines are overflow content

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

## 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

--> The overflow content will be hidden from the user

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

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

# overflow: auto

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

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

# 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 {  
  
  background-color: #eee;  
  
  width: 200px;  
  
  height: 50px;
```



```
border: 1px dotted black;
```

```
overflow-x: hidden;
```

```
overflow-y: scroll;
```

```
}
```

# 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.

## 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...



### Example

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

## 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...

## Example

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

## 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...

## Example

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

# The clear Property

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

The **clear** property can have one of the following values:

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

The most common way to use the `clear` property is after you have used a `float` property on an element.

When clearing floats, you should match the clear to the float: If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.

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

# Layout - Float Examples

## Grid of Boxes / Equal Width Boxes



With the `float` property, it is easy to float boxes of content side by side:

```
* {  
  box-sizing: border-box;  
}
```

```
.box {  
  float: left;  
  width: 33.33%;  
  padding: 50px;  
}
```

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

## Equal Height Boxes

In the previous example, you learned how to float boxes side by side with an equal width. However, it is not easy to create floating boxes with equal heights. A quick fix however, is to set a fixed height, like in the example below:



### Example

```
.box {  
  height: 500px;  
}
```

## 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`:

```
span.b {  
  display: inline-block;  
  width: 100px;  
  height: 100px;  
  padding: 5px;  
  border: 1px solid blue;  
  background-color: yellow;  
}
```

# 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:

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  margin: auto;
  width: 60%;
  border: 3px solid #73AD21;
  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><b>Note: </b>Using margin:auto will not work in IE8, unless a !DOCTYPE
is declared.</p>
</div>
```

### Center Align Elements

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

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

## 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;
}
```

Try it Yourself »

## Center an Image

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

```
img {
```

```
  display: block;
```

```
  margin-left: auto;
```

```
margin-right: auto;
}
```

## Left and Right Align - Using position

One method for aligning elements is to use `position: absolute;`: By using these process we can float the element on right side

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

--> We can also use flote property to float the element on right side like `float:right`

## 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:

```
/* 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;  
}
```

**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 CSS Classes

Pseudo-classes can be combined with CSS classes:

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

```
a.highlight:hover {  
    color: #ff0000;  
}  
</style>  
</head>  
<body>
```

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

```
</body>
```

## Hover on <div>

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

```
div {
  background-color: green;
  color: white;
  padding: 25px;
  text-align: center;
}
```

```
div:hover {
  background-color: blue;
}
```

## Simple Tooltip Hover

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

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

```
div:hover p {
  display: block;
}
```

## 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:

```
<!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>
<p><b>Note:</b> For :first-child to work in IE8 and earlier, a DOCTYPE must be
declared.</p>
```

This is some text.

This is some text.

**Note:** For :first-child to work in IE8 and earlier, a DOCTYPE must be declared.

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



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

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

Result:

I am a *strong* person. I am a *strong* person.

I am a *strong* person. I am a *strong* person.

**Note:** For `:first-child` to work in IE8 and earlier, a DOCTYPE must be declared.

# Pseudo-elements

## 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;  
}
```

--> Which makes the first to #ff0000 color

## The ::first-letter Pseudo-element

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

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

## Pseudo-elements and CSS Classes

Pseudo-elements can be combined with CSS classes:

```
p.intro::first-letter {  
  
  color: #ff0000;  
  
  font-size: 200%;  
}
```

## 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:

## 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);  
}
```

## 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:

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

## 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:

```
::-moz-selection { /* Code for Firefox */  
  color: red;  
  background: yellow;  
}  
  
::selection {  
  color: red;  
  background: yellow;  
}
```

# Vertical Navigation Bar

```
<style>
```

```
body {
```

```
margin: 0;
```

```
}
```

```
ul {
```

```
list-style-type: none;
```

```
margin: 0;
```

```
padding: 0;
```

```
width: 25%;
```

```
background-color: #f1f1f1;
```

```
position: fixed;
```

```
height: 100%;
```

```
overflow: auto;
```

```
}
```

```
li a {
```

```
display: block;
```

```
color: #000;
```

```
padding: 8px 16px;
```

```
text-decoration: none;
```

```
}
```

```
li a.active {
```

```
background-color: #4CAF50;

color: white;

}
```

```
li a:hover:not(.active) {

background-color: #555;

color: white;

}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<ul>
```

```
<li><a class="active" href="#home">Home</a></li>
```

```
<li><a href="#news">News</a></li>
```

```
<li><a href="#contact">Contact</a></li>
```

```
<li><a href="#about">About</a></li>
```

```
</ul>
```

```
<div style="margin-left:25%;padding:1px 16px;height:1000px;">
```

```
<h2>Fixed Full-height Side Nav</h2>
```

```
<h3>Try to scroll this area, and see how the sidenav sticks to the
page</h3>
```

```
<p>Notice that this div element has a left margin of 25%. This is because
the side navigation is set to 25% width. If you remove the margin, the
sidenav will overlay/sit on top of this div.</p>
```

```
<p>Also notice that we have set overflow:auto to sidenav. This will add a
scrollbar when the sidenav is too long (for example if it has over 50 links
inside of it).</p>
```

```
<p>Some text..</p>
<p>Some text..</p>
<p>Some text..</p>
<p>Some text..</p>
<p>Some text..</p>
<p>Some text..</p>
<p>Some text..</p>
</div>
```

```
</body>
```

# Horizontal Navigation Bar

```
<style>

ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

li a:hover:not(.active) {
  background-color: #111;
}

.active {
```

```

background-color: #4CAF50;
}
</style>
</head>
<body>

<ul>
  <li><a class="active" href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li><a href="#about">About</a></li>
</ul>

</body>

```

# Dropdowns

## Basic Dropdown

Create a dropdown box that appears when the user moves the mouse over an element.

```

<style>
.dropdown {
  position: relative;
  display: inline-block;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  padding: 12px 16px;
  z-index: 1;
}

.dropdown:hover .dropdown-content {
  display: block;
}
</style>
</head>
<body>

<h2>Hoverable Dropdown</h2>
<p>Move the mouse over the text below to open the dropdown content.</p>

<div class="dropdown">
  <span>Mouse over me</span>
  <div class="dropdown-content">
    <p>Hello World!</p>
  </div>
</div>

```

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

## Example Explained

**HTML)** Use any element to open the dropdown content, e.g. a `<span>`, or a `<button>` element.

Use a container element (like `<div>`) to create the dropdown content and add whatever you want inside of it.

Wrap a `<div>` element around the elements to position the dropdown content correctly with CSS.

**CSS)** The `.dropdown` class uses `position:relative`, which is needed when we want the dropdown content to be placed right below the dropdown button (using `position:absolute`).

The `.dropdown-content` class holds the actual dropdown content. It is hidden by default, and will be displayed on hover (see below). Note the `min-width` is set to 160px. Feel free to change this. **Tip:** If you want the width of the dropdown content to be as wide as the dropdown button, set the `width` to 100% (and `overflow:auto` to enable scroll on small screens).

Instead of using a border, we have used the CSS `box-shadow` property to make the dropdown menu look like a "card".

The `:hover` selector is used to show the dropdown menu when the user moves the mouse over the dropdown button.

```
<style>  
.dropbtn {  
  background-color: #4CAF50;  
  color: white;  
  padding: 16px;  
  font-size: 16px;  
  border: none;  
  cursor: pointer;  
}  
  
.dropdown {  
  position: relative;  
  display: inline-block;  
}  
  
.dropdown-content {  
  display: none;  
  position: absolute;  
  background-color: #f9f9f9;  
  min-width: 160px;  
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);  
}
```



```
z-index: 1;
}
```

```
.dropdown-content a {
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
}
```

```
.dropdown-content a:hover {background-color: #f1f1f1}
```

```
.dropdown:hover .dropdown-content {
  display: block;
}
```

```
.dropdown:hover .dropbtn {
  background-color: #3e8e41;
}
```

```
</style>
</head>
<body>
```

```
<h2>Dropdown Menu</h2>
```

```
<p>Move the mouse over the button to open the dropdown menu.</p>
```

```
<div class="dropdown">
  <button class="dropbtn">Dropdown</button>
  <div class="dropdown-content">
    <a href="#">Link 1</a>
    <a href="#">Link 2</a>
    <a href="#">Link 3</a>
  </div>
</div>
```

```
<p><strong>Note:</strong> We use href="#" for test links. In a real web
site this would be URLs.</p>
```

```
</body>
```

# Attribute Selectors

## Style HTML Elements With Specific Attributes

It is possible to style HTML elements that have specific attributes or attribute values.

---

## CSS [attribute] Selector

The `[attribute]` selector is used to select elements with a specified attribute.

The following example selects all `<a>` elements with a target attribute:

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

## CSS [attribute="value"] Selector

The `[attribute="value"]` selector is used to select elements with a specified attribute and value.

The following example selects all `<a>` elements with a target="\_blank" attribute:

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

## CSS [attribute]="value" Selector

The `[attribute]="value"` selector is used to select elements with the specified attribute starting with the specified value.

The following example selects all elements with a class attribute value that begins with "top":

**Note:** The value has to be a whole word, either alone, like class="top", or followed by a hyphen( - ), like class="top-text"!

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

## CSS [attribute^="value"] Selector

The `[attribute^="value"]` selector is used to select elements whose attribute value begins with a specified value.

The following example selects all elements with a class attribute value that begins with "top":

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

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

## CSS [attribute\$="value"] Selector

The `[attribute$="value"]` selector is used to select elements whose attribute value ends with a specified value.

The following example selects all elements with a class attribute value that ends with "test":

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

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

## [attribute\*="value"] Selector

The `[attribute*="value"]` selector is used to select elements whose attribute value contains a specified value.

The following example selects all elements with a class attribute value that contains "te":

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

```
[class*="te"] {  
  background: yellow;  
}
```

# Forms

## Styling Input Fields

Use the `width` property to determine the width of the input field:

```
input {  
  width: 100%;  
}
```

The example above applies to all `<input>` elements. If you only want to style a specific input type, you can use attribute selectors:

- `input[type=text]` - will only select text fields
- `input[type=password]` - will only select password fields
- `input[type=number]` - will only select number fields
- etc..

## Padded Inputs

Use the `padding` property to add space inside the text field.

**Tip:** When you have many inputs after each other, you might also want to add some `margin`, to add more space outside of them:

```
input[type=text] {  
  width: 100%;  
  padding: 12px 20px;  
  margin: 8px 0;  
  box-sizing: border-box;  
}
```

## Bordered Inputs

Use the `border` property to change the border size and color, and use the `border-radius` property to add rounded corners:

```
input[type=text] {  
  width: 100%;  
  padding: 12px 20px;  
  margin: 8px 0;  
  box-sizing: border-box;  
  border: 2px solid red;  
  border-radius: 4px;  
}
```

## Focused Inputs

By default, some browsers will add a blue outline around the input when it gets focus (clicked on). You can remove this behavior by adding `outline: none;` to the input.

Use the `:focus` selector to do something with the input field when it gets focus:

```
input[type=text] {  
  width: 100%;  
  padding: 12px 20px;  
  margin: 8px 0;  
  box-sizing: border-box;  
}
```

```
border: 1px solid #555;  
outline: none;  
}
```

## Input with icon/image

If you want an icon inside the input, use the `background-image` property and position it with the `background-position` property. Also notice that we add a large left padding to reserve the space of the icon:



```
input[type=text] {  
  width: 100%;  
  box-sizing: border-box;  
  border: 2px solid #ccc;  
  border-radius: 4px;  
  font-size: 16px;  
  background-color: white;  
  background-image: url('searchicon.png');  
  background-position: 10px 10px;  
  background-repeat: no-repeat;  
  padding: 12px 20px 12px 40px;  
}
```

## Animated Search Input

In this example we use the CSS `transition` property to animate the width of the search input when it gets focus. You will learn more about the `transition` property later, in our [CSS Transitions](#) chapter.

```
input[type=text] {  
  
  width: 130px;  
  
  box-sizing: border-box;  
  
  border: 2px solid #ccc;  
  
  border-radius: 4px;  
  
  font-size: 16px;  
  
  background-color: white;  
  
  background-image: url('searchicon.png');  
  
  background-position: 10px 10px;
```

```
background-repeat: no-repeat;

padding: 12px 20px 12px 40px;

transition: width 0.4s ease-in-out;

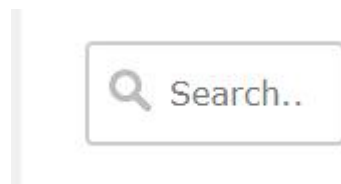
}
```

```
input[type=text]:focus {

    width: 100%;

}
```

Before click on input box:



After clicking on input box



## Styling Input Buttons

```
input[type=button], input[type=submit], input[type=reset] {
    background-color: #4CAF50;
    border: none;
    color: white;
    padding: 16px 32px;
    text-decoration: none;
    margin: 4px 2px;
    cursor: pointer;
}
```

## Responsive Form

Resize the browser window to see the effect. When the screen is less than 600px wide, make the two columns stack on top of each other instead of next to each other.

**Advanced:** The following example use [media queries](#) to create a responsive form. You will learn more about this in a later chapter.

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
    box-sizing: border-box;

input[type=text], select, textarea {
    width: 100%;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    resize: vertical;
}

label {
    padding: 12px 12px 12px 0;
    display: inline-block;
}

input[type=submit] {
    background-color: #4CAF50;
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    float: right;
}

input[type=submit]:hover {
    background-color: #45a049;
}

.container {
    border-radius: 5px;
    background-color: #f2f2f2;
    padding: 20px;
}

.col-25 {
    float: left;
    width: 25%;
    margin-top: 6px;
}

.col-75 {
    float: left;
    width: 75%;
    margin-top: 6px;
}
```

```
/* Clear floats after the columns */
```

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

```
/* Responsive layout - when the screen is less than 600px wide, make the  
two columns stack on top of each other instead of next to each other */
```

```
@media screen and (max-width: 600px) {  
  .col-25, .col-75, input[type=submit] {  
    width: 100%;  
    margin-top: 0;  
  }  
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h2>Responsive Form</h2>
```

```
<p>Resize the browser window to see the effect. When the screen is less  
than 600px wide, make the two columns stack on top of each other instead  
of next to each other.</p>
```

```
<div class="container">
```

```
  <form action="/action_page.php">
```

```
    <div class="row">
```

```
      <div class="col-25">
```

```
        <label for="fname">First Name</label>
```

```
      </div>
```

```
      <div class="col-75">
```

```
        <input type="text" id="fname" name="firstname" placeholder="Your  
name..">
```

```
      </div>
```

```
    </div>
```

```
    <div class="row">
```

```
      <div class="col-25">
```

```
        <label for="lname">Last Name</label>
```

```
      </div>
```

```
      <div class="col-75">
```

```
        <input type="text" id="lname" name="lastname" placeholder="Your  
last name..">
```

```
      </div>
```

```
    </div>
```

```
    <div class="row">
```

```
      <div class="col-25">
```

```
        <label for="country">Country</label>
```

```
      </div>
```

```
      <div class="col-75">
```

```
        <select id="country" name="country">
```

```
          <option value="australia">Australia</option>
```



```

        <option value="canada">Canada</option>
        <option value="usa">USA</option>
    </select>
</div>
</div>
<div class="row">
    <div class="col-25">
        <label for="subject">Subject</label>
    </div>
    <div class="col-75">
        <textarea id="subject" name="subject" placeholder="Write
something.." style="height:200px"></textarea>
    </div>
</div>
<div class="row">
    <input type="submit" value="Submit">
</div>
</form>
</div>

</body>
</html>

```

## Responsive Website Layout

By using some of the CSS code above, we have created a responsive website layout, which varies between two columns and full-width columns depending on screen width:

```

<!DOCTYPE html>
<html>
<head>
<style>
* {
    box-sizing: border-box;
}

body {
    font-family: Arial;
    padding: 10px;
    background: #f1f1f1;
}

/* Header/Blog Title */
.header {
    padding: 30px;
    text-align: center;
    background: white;
}

.header h1 {

```

```
font-size: 50px;
}

/* Style the top navigation bar */
.topnav {
  overflow: hidden;
  background-color: #333;
}

/* Style the topnav links */
.topnav a {
  float: left;
  display: block;
  color: #f2f2f2;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

/* Change color on hover */
.topnav a:hover {
  background-color: #ddd;
  color: black;
}

/* Create two unequal columns that floats next to each other */
/* Left column */
.leftcolumn {
  float: left;
  width: 75%;
}

/* Right column */
.rightcolumn {
  float: left;
  width: 25%;
  background-color: #f1f1f1;
  padding-left: 20px;
}

/* Fake image */
.fakeimg {
  background-color: #aaa;
  width: 100%;
  padding: 20px;
}

/* Add a card effect for articles */
.card {
  background-color: white;
  padding: 20px;
  margin-top: 20px;
}
```

```
}
```

```
/* Clear floats after the columns */
```

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

```
/* Footer */
```

```
.footer {  
  padding: 20px;  
  text-align: center;  
  background: #ddd;  
  margin-top: 20px;  
}
```

```
/* Responsive layout - when the screen is less than 800px wide, make the  
two columns stack on top of each other instead of next to each other */
```

```
@media screen and (max-width: 800px) {  
  .leftcolumn, .rightcolumn {  
    width: 100%;  
    padding: 0;  
  }  
}
```

```
/* Responsive layout - when the screen is less than 400px wide, make the  
navigation links stack on top of each other instead of next to each other */
```

```
@media screen and (max-width: 400px) {  
  .topnav a {  
    float: none;  
    width: 100%;  
  }  
}
```

```
</style>  
</head>  
<body>
```

```
<div class="header">  
  <h1>My Website</h1>  
  <p>Resize the browser window to see the effect.</p>  
</div>
```

```
<div class="topnav">  
  <a href="#">Link</a>  
  <a href="#">Link</a>  
  <a href="#">Link</a>  
  <a href="#" style="float:right">Link</a>  
</div>
```

```
<div class="row">  
  <div class="leftcolumn">
```

```

    <div class="card">
      <h2>TITLE HEADING</h2>
      <h5>Title description, Dec 7, 2017</h5>
      <div class="fakeimg" style="height:200px;">Image</div>
      <p>Some text..</p>
      <p>Sunt in culpa qui officia deserunt mollit anim id est laborum
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco.</p>
    </div>
    <div class="card">
      <h2>TITLE HEADING</h2>
      <h5>Title description, Sep 2, 2017</h5>
      <div class="fakeimg" style="height:200px;">Image</div>
      <p>Some text..</p>
      <p>Sunt in culpa qui officia deserunt mollit anim id est laborum
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco.</p>
    </div>
  </div>
  <div class="rightcolumn">
    <div class="card">
      <h2>About Me</h2>
      <div class="fakeimg" style="height:100px;">Image</div>
      <p>Some text about me in culpa qui officia deserunt mollit
anim..</p>
    </div>
    <div class="card">
      <h3>Popular Post</h3>
      <div class="fakeimg"><p>Image</p></div>
      <div class="fakeimg"><p>Image</p></div>
      <div class="fakeimg"><p>Image</p></div>
    </div>
    <div class="card">
      <h3>Follow Me</h3>
      <p>Some text..</p>
    </div>
  </div>
</div>

<div class="footer">
  <h2>Footer</h2>
</div>

</body>
</html>

```

# Rounded Corners

## CSS border-radius - Specify Each Corner

The `border-radius` property can have from one to four values. Here are the rules:

**Four values - `border-radius: 15px 50px 30px 5px`;** (first value applies to top-left corner, second value applies to top-right corner, third value applies to bottom-right corner, and fourth value applies to bottom-left corner):



**Three values - `border-radius: 15px 50px 30px`;** (first value applies to top-left corner, second value applies to top-right and bottom-left corners, and third value applies to bottom-right corner):



**Two values - `border-radius: 15px 50px`;** (first value applies to top-left and bottom-right corners, and the second value applies to top-right and bottom-left corners):



**One value - `border-radius: 15px`;** (the value applies to all four corners, which are rounded equally):



Here is the code:

# Multiple Backgrounds

In this chapter you will learn how to add multiple background images to one element.

You will also learn about the following properties:

- `background-size`
- `background-origin`
- `background-clip`



# Multiple Backgrounds

CSS allows you to add multiple background images for an element, through the `background-image` property.

The different background images are separated by commas, and the images are stacked on top of each other, where the first image is closest to the viewer.

The following example has two background images, the first image is a flower (aligned to the bottom and right) and the second image is a paper background (aligned to the top-left corner):

## Example

```
#example1 {  
  background-image: url(img_flwr.gif), url(paper.gif);  
  background-position: right bottom, left top;  
  background-repeat: no-repeat, repeat;  
  padding: 15px;  
}
```

Multiple background images can be specified using either the individual background properties (as above) or the `background` shorthand property.

The following example uses the `background` shorthand property (same result as example above):

## Example

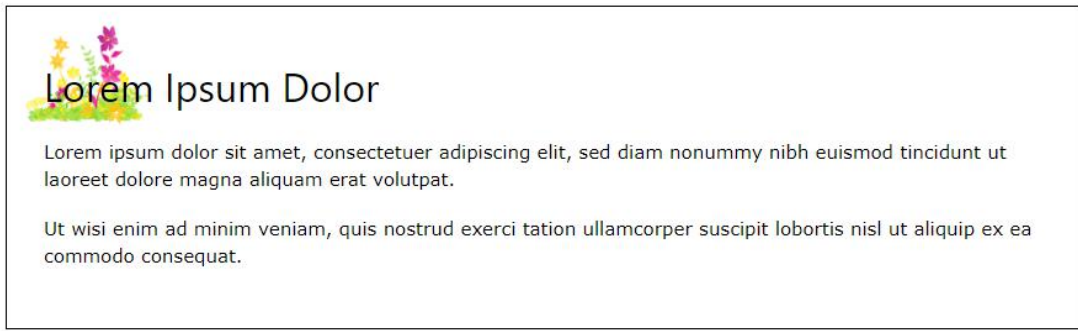
```
#example1 {  
  background: url(img_flwr.gif) right bottom no-repeat, url(paper.gif) left  
top repeat;  
  padding: 15px;  
}
```

# Background Size

The CSS `background-size` property allows you to specify the size of background images.

The size can be specified in lengths, percentages, or by using one of the two keywords: contain or cover.

```
#example1 {  
  border: 1px solid black;  
  background:url(img_flwr.gif);  
  background-size: 100px 80px;  
  background-repeat: no-repeat;  
  padding:15px;  
}
```



The two other possible values for `background-size` are `contain` and `cover`.

The `contain` keyword scales the background image to be as large as possible (but both its width and its height must fit inside the content area). As such, depending on the proportions of the background image and the background positioning area, there may be some areas of the background which are not covered by the background image.

The `cover` keyword scales the background image so that the content area is completely covered by the background image (both its width and height are equal to or exceed the content area). As such, some parts of the background image may not be visible in the background positioning area.

The following example illustrates the use of `contain` and `cover`:

```
.div1 {  
  border: 1px solid black;  
  height:120px;  
  width:150px;  
  background:url(img_flwr.gif);  
  background-repeat: no-repeat;  
  background-size: contain;  
}
```

# The background-size Property

**background-size: contain:**



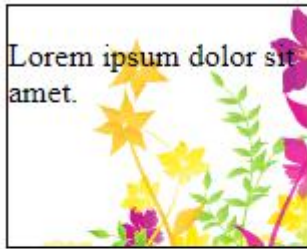
**background-size: cover:**



```
.div2 {  
  border: 1px solid black;  
  height: 120px;  
  width: 150px;  
  background: url(img_flwr.gif);  
  background-repeat: no-repeat;  
  background-size: cover;  
}
```



## No background-size defined:



Original image:



```
.div3 {  
  border: 1px solid black;  
  height: 120px;  
  width: 150px;  
  background: url(img_flwr.gif);  
  background-repeat: no-repeat;  
}
```

## Full Size Background Image

Now we want to have a background image on a website that covers the entire browser window at all times.

The requirements are as follows:

- Fill the entire page with the image (no white space)
- Scale image as needed
- Center image on page
- Do not cause scrollbars

The following example shows how to do it; Use the `<html>` element (the `<html>` element is always at least the height of the browser window). Then set a fixed and centered background on it. Then adjust its size with the `background-size` property:

```
html {
```

```
background: url(img_man.jpg) no-repeat center fixed;  
background-size: cover;  
}
```

## CSS background-origin Property

The CSS `background-origin` property specifies where the background image is positioned.

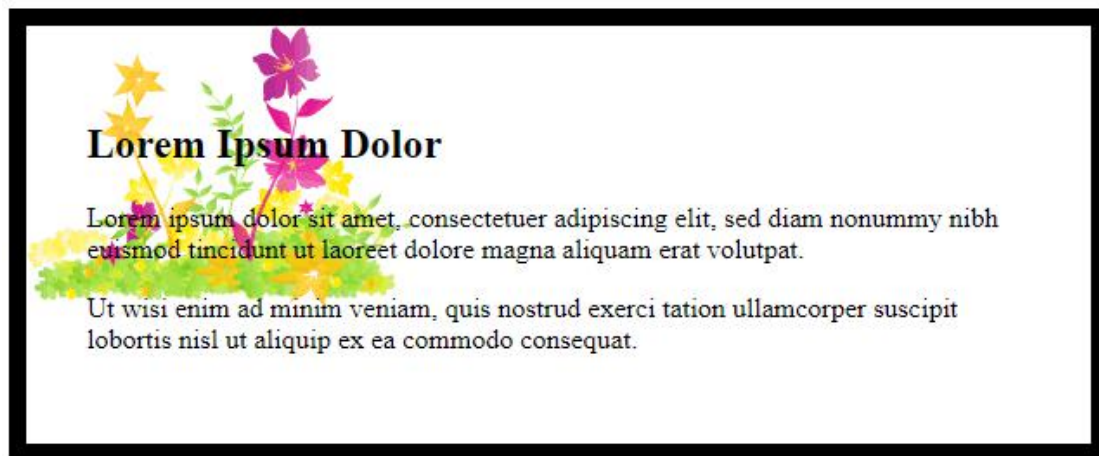
The property takes three different values:

- border-box - the background image starts from the upper left corner of the border
- padding-box - (default) the background image starts from the upper left corner of the padding edge
- content-box - the background image starts from the upper left corner of the content

The following example illustrates the `background-origin` property:

```
#example1 {  
  border: 10px solid black;  
  padding: 35px;  
  background: url(img_flwr.gif);  
  background-repeat: no-repeat;  
}
```

No background-origin (padding-box is default):



```
#example2 {  
  border: 10px solid black;  
  padding: 35px;  
  background: url(img_flwr.gif);  
  background-repeat: no-repeat;  
  background-origin: border-box;  
}
```

```
}
```

```
background-origin: border-box;
```



```
#example3 {
```

```
border: 10px solid black;
```

```
padding: 35px;
```

```
background: url(img_flwr.gif);
```

```
background-repeat: no-repeat;
```

```
background-origin: content-box;
```

```
}
```

```
background-origin: content-box;
```



## CSS background-clip Property

The CSS `background-clip` property specifies the painting area of the background.

The property takes three different values:

- `border-box` - (default) the background is painted to the outside edge of the border
- `padding-box` - the background is painted to the outside edge of the padding
- `content-box` - the background is painted within the content box

The following example illustrates the `background-clip` property:

```
#example1 {  
  border: 10px dotted black;  
  padding: 35px;  
  background: yellow;  
}
```

No background-clip (border-box is default):



```
#example2 {  
  border: 10px dotted black;  
  padding: 35px;  
  background: yellow;  
  background-clip: padding-box;  
}
```

background-clip: padding-box:



```
#example3 {  
  border: 10px dotted black;  
  padding: 35px;  
  background: yellow;  
  background-clip: content-box;  
}
```

}

background-clip: content-box;



# Gradients

CSS gradients let you display smooth transitions between two or more specified colors.

CSS defines two types of gradients:

- **Linear Gradients (goes down/up/left/right/diagonally)**
- **Radial Gradients (defined by their center)**

---

## CSS Linear Gradients

To create a linear gradient you must define at least two color stops. Color stops are the colors you want to render smooth transitions among. You can also set a starting point and a direction (or an angle) along with the gradient effect.

### Syntax

```
background-image:  
linear-gradient(direction, color-stop1, color-stop2, ...);
```

## Linear Gradient -

The following example shows a linear gradient that starts at the top. It starts red, transitioning to yellow:



### Example

```
#grad {  
  background-image: linear-gradient(red, yellow);  
}
```

[Try it Yourself >](#)

## Top to Bottom (this is default)

### Linear Gradient - Diagonal

You can make a gradient diagonally by specifying both the horizontal and vertical starting positions.

The following example shows a linear gradient that starts at top left (and goes to bottom right). It starts red, transitioning to yellow:



### Example

```
#grad {  
  background-image: linear-gradient(to bottom right, red, yellow);  
}
```

### Linear Gradient - Left to Right

The following example shows a linear gradient that starts from the left. It starts red, transitioning to yellow:



### Example

```
#grad {  
  background-image: linear-gradient(to right, red , yellow);  
}
```

[Try it Yourself >](#)

# Using Angles

If you want more control over the direction of the gradient, you can define an angle, instead of the predefined directions (to bottom, to top, to right, to left, to bottom right, etc.).

## Syntax

```
background-image: linear-gradient(angle, color-stop1, color-stop2);
```

The angle is specified as an angle between a horizontal line and the gradient line.

The following example shows how to use angles on linear gradients:



## Example

```
#grad {  
  background-image: linear-gradient(-90deg, red, yellow);  
}
```

Screenshots

# Using Multiple Color Stops

The following example shows a linear gradient (from top to bottom) with multiple color stops:



## Example

```
#grad {  
  background-image: linear-gradient(red, yellow, green);  
}
```

Twit Yourself



## Using Transparency

CSS gradients also support transparency, which can be used to create fading effects.

To add transparency, we use the `rgba()` function to define the color stops. The last parameter in the `rgba()` function can be a value from 0 to 1, and it defines the transparency of the color: 0 indicates full transparency, 1 indicates full color (no transparency).

The following example shows a linear gradient that starts from the left. It starts fully transparent, transitioning to full color red:



### Example

```
#grad {  
  background-image: linear-gradient(to right, rgba(255,0,0,0), rgba(255,0,0,1));  
}
```

## Repeating a linear-gradient

The `repeating-linear-gradient()` function is used to repeat linear gradients:



### Example

A repeating linear gradient:

```
#grad {  
  background-image: repeating-linear-gradient(red, yellow 10%, green 20%);  
}
```

[Try it Yourself »](#)



The following example shows how to create a linear gradient (from left to right) with the color of the rainbow and some text:



#### Example

```
#grad {  
  background-image: linear-gradient(to right, red,orange,yellow,green,blue,indigo,violet);  
}
```

[Try it Yourself »](#)

## CSS Radial Gradients

A radial gradient is defined by its center.

To create a radial gradient you must also define at least two color stops.

### Syntax

---

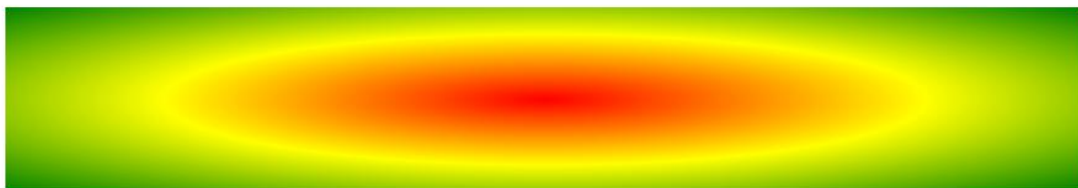
#### Syntax

```
background-image: radial-gradient(shape size at position, start-color, ..., last-color);
```

By default, shape is ellipse, size is farthest-corner, and position is center.

#### Radial Gradient - Evenly Spaced Color Stops (this is default)

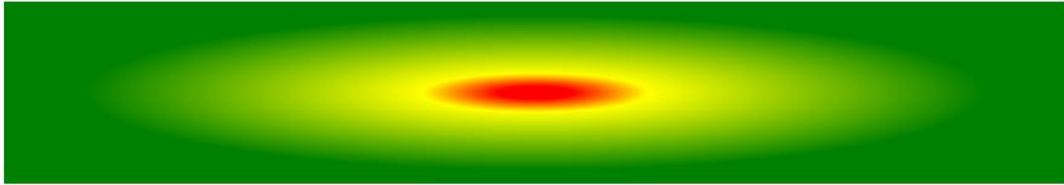
The following example shows a radial gradient with evenly spaced color stops:



```
#grad1 {  
  height: 150px;  
  width: 200px;  
  background-color: red; /* For browsers that do not support gradients */  
  background-image: radial-gradient(red, yellow, green); /* Standard  
syntax (must be last) */  
}
```

### Radial Gradient - Differently Spaced Color Stops

The following example shows a radial gradient with differently spaced color stops:



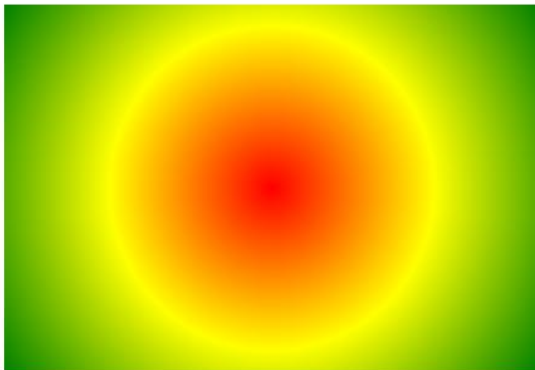
#### Example

```
#grad {  
  background-image: radial-gradient(red 5%, yellow 15%, green 60%);  
}
```

## Set Shape

The shape parameter defines the shape. It can take the value circle or ellipse. The default value is ellipse.

The following example shows a radial gradient with the shape of a circle:

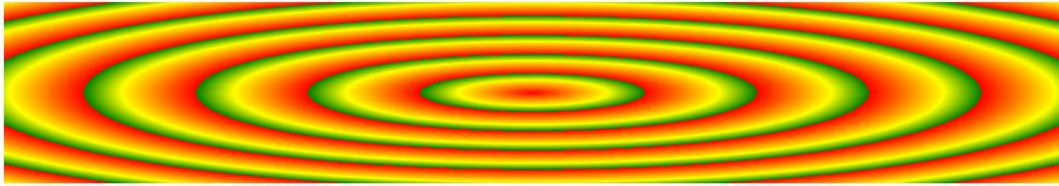


#### Example

```
#grad {  
  background-image: radial-gradient(circle, red, yellow, green);  
}
```

## Repeating a radial-gradient

The `repeating-radial-gradient()` function is used to repeat radial gradients:



### Example

A repeating radial gradient:

```
#grad {  
  background-image: repeating-radial-gradient(red, yellow 10%, green 15%);  
}
```

# ShadowEffects



Box Shadow

With CSS you can create  
shadow effects!

Hover over me!

## CSS Shadow Effects

With CSS you can add shadow to text and to elements.

In this chapter you will learn about the following properties:

- `text-shadow`
- `box-shadow`

## Text shadow effect!

### Example

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

[Try it Yourself »](#)

Next, add a color to the shadow:

## Text shadow effect!

### Example

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

## Multiple Shadows

To add more than one shadow to the text, you can add a comma-separated list of shadows.

The following example shows a red and blue neon glow shadow:

## Text shadow effect!

### Example

```
h1 {  
  text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF;  
}
```

You can also use the text-shadow property to create a plain border around some text (without shadows):

## Border around text!

### Example

```
h1 {  
  color: yellow;  
  text-shadow: -1px 0 black, 0 1px black, 1px 0 black, 0 -1px black;  
}
```

# CSS box-shadow Property

The CSS `box-shadow` property applies shadow to elements.

In its simplest use, you only specify the horizontal shadow and the vertical shadow:

This is a yellow <div> element  
with a black box-shadow

## Example

```
div {  
  box-shadow: 10px 10px;  
}
```

Next, add a color to the shadow:

This is a yellow <div> element  
with a grey box-shadow

## Example

```
div {  
  box-shadow: 10px 10px grey;  
}
```

Next, add a blur effect to the shadow:

This is a yellow <div> element  
with a blurred, grey box-shadow

## Example

```
div {  
  box-shadow: 10px 10px 5px grey;  
}
```

Note: If we want the shadow around the box then use this syntax

4 sides of Box:

Syntax:

```
div{  
Box-shadow: 10px 10px 5px grey,-10px -10px 5px grey;  
}
```

# 2D Transforms

CSS transforms allow you to move, rotate, scale, and skew elements.

Mouse over the element below to see a 2D transformation:

## Transforms Methods

With the CSS `transform` property you can use the following 2D transformation methods:

- `translate()`
- `rotate()`
- `scaleX()`
- `scaleY()`
- `scale()`
- `skewX()`
- `skewY()`
- `skew()`
- `matrix()`

### The `translate()` Method



The `translate()` method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).

The following example moves the `<div>` element 50 pixels to the right, and 100 pixels down from its current position:

#### Example

```
div {  
  transform: translate(50px, 100px);  
}
```

## The rotate() Method



The `rotate()` method rotates an element clockwise or counter-clockwise according to a given degree.

The following example rotates the `<div>` element clockwise with 20 degrees:

### Example

```
div {  
  transform: rotate(20deg);  
}
```

## The scale() Method



The `scale()` method increases or decreases the size of an element (according to the parameters given for the width and height).

The following example increases the `<div>` element to be two times of its original width, and three times of its original height:

### Example

```
div {  
  transform: scale(2, 3);  
}
```

# The scaleX() Method

The `scaleX()` method increases or decreases the width of an element.

The following example increases the `<div>` element to be two times of its original width:

```
div {  
  margin: 150px;  
  width: 200px;  
  height: 100px;  
  background-color: yellow;  
  border: 1px solid black;  
  -ms-transform: scaleX(2); /* IE 9 */  
  transform: scaleX(2); /* Standard syntax */  
}
```

The following example decreases the <div> element to be half of its original width: `transform: scaleX(0.5);`

## The scaleY() Method

The `scaleY()` method increases or decreases the height of an element.

The following example increases the <div> element to be three times of its original height:

```
div{
transform: scaleY(3); /* Standard syntax */
}
```

The following example decreases the <div> element to be half of its original Height: `transform: scaleY(0.5);`

## The skew() Method

The `skew()` method skews an element along the X and Y-axis by the given angles.

The following example skews the <div> element 20 degrees along the X-axis, and 10 degrees along the Y-axis:

```
div {
width: 300px;
height: 100px;
background-color: yellow;
border: 1px solid black;
}
```

```
div#myDiv {
transform: skew(20deg,10deg); /* Standard syntax */
}
```

If the second parameter is not specified, it has a zero value. So, the following example skews the <div> element 20 degrees along the X-axis:

```
div#myDiv {
transform: skew(20deg); /* Standard syntax */
}
```



## The matrix() Method



The `matrix()` method combines all the 2D transform methods into one.

The `matrix()` method takes six parameters, containing mathematical functions, which allows you to rotate, scale, move (translate), and skew elements.

The parameters are as follows: `matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY())`

### Example

```
div {  
  transform: matrix(1, -0.3, 0, 1, 0, 0);  
}
```

# 3D Transforms

With the CSS `transform` property you can use the following 3D transformation methods:

- `rotateX()`
- `rotateY()`
- `rotateZ()`

## The rotateX() Method

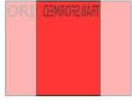


The `rotateX()` method rotates an element around its X-axis at a given degree:

### Example

```
#myDiv {  
  transform: rotateX(150deg);  
}
```

## The rotateY() Method



The `rotateY()` method rotates an element around its Y-axis at a given degree:

### Example

```
#myDiv {  
  transform: rotateY(130deg);  
}
```

## The rotateZ() Method

The `rotateZ()` method rotates an element around its Z-axis at a given degree:

### Example

```
#myDiv {  
  transform: rotateZ(90deg);  
}
```

# Transitions

## CSS Transitions

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

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:

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

## 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 {  
  transition: width 2s, height 4s;  
}
```

## 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 the some of the different speed curves that can be used:

```
#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:

```
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);  
}
```

SHORT CUT FOR `transition`:

```
div {  
  transition: width 2s linear 1s;  
}
```

## CSS Animations

CSS allows animation of HTML elements without using JavaScript or Flash!

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 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;
}
```

**Note:** The `animation-duration` property defines how long time 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:

## Example

```
/* 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;
}
```

## 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:

## Example

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

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;  
}
```

## 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:

## Example

```
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:

## Example

```
div {  
  width: 100px;
```

```
height: 100px;  
position: relative;  
background-color: red;  
animation-name: example;  
animation-duration: 4s;  
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):

### Example

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

## 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 the 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:

## Example

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

## Animation Shorthand Property

The example below uses six of the animation properties:

## Example

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

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

## Example

```
div {  
  animation: example 5s linear 2s infinite alternate;  
}
```