https://www.w3schools.com/

CSS Tutorial

CSS is a language that describes the style of an HTML document.

CSS describes how HTML elements should be displayed.

This tutorial will teach you CSS from basic to advanced.

CSS Example

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

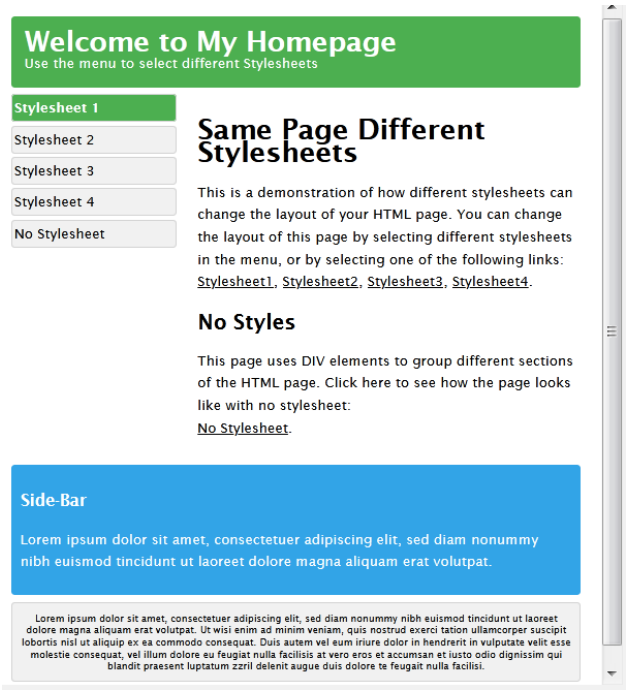
CSS Introduction

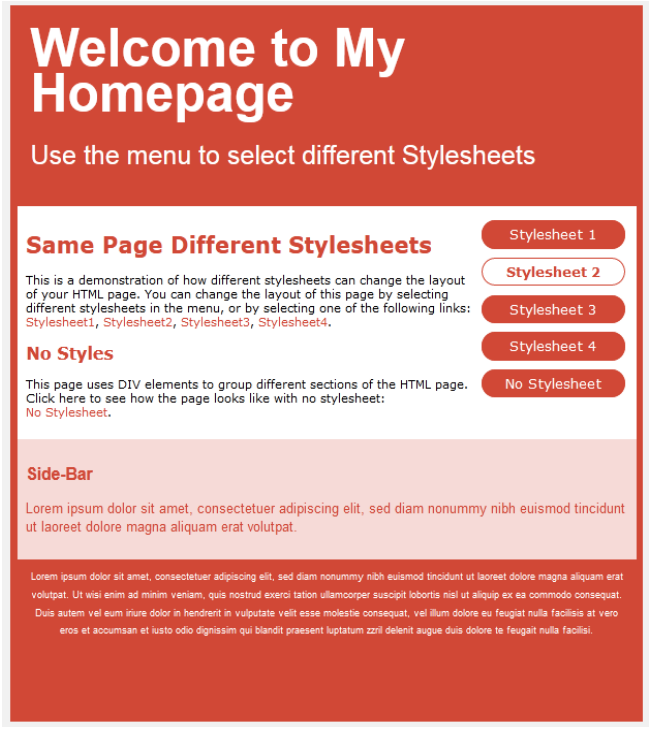
What is CSS?

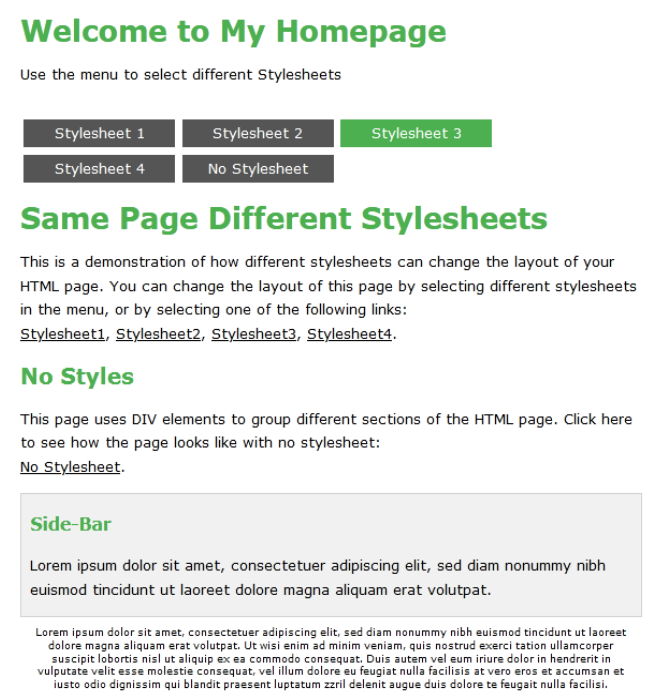
- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**

CSS Demo - One HTML Page - Multiple Styles!

Here we will show one HTML page displayed with four different stylesheets. Click on the "Stylesheet 1", "Stylesheet 2", "Stylesheet 3", "Stylesheet 4" links below to see the different styles:







Why 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.

CSS Solved a Big Problem

HTML was NEVER intended to contain tags for formatting a web page!

HTML was created to **describe the content** of a web page, like:

<h1>This is a heading</h1>

This is a paragraph.

When tags like <font>, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large websites, where fonts and color information were added to every single page, became a long and expensive process.

To solve this problem, the World Wide Web Consortium (W3C) created CSS.

CSS removed the style formatting from the HTML page!

CSS Saves a Lot of Work!

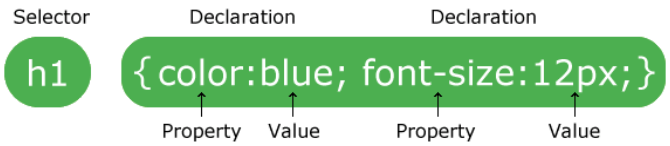
The style definitions are normally saved in external .css files.

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

CSS Syntax

CSS 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

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

```
p {
  color: red;
  text-align: center;
}
```

Example Explained

- **p** is a **selector** in CSS (it points to the HTML element you want to style: <p>).
- **color** is a property, and **red** is the property value
- **text-align** is a property, and **center** is the property value

CSS Selectors

CSS Selectors

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

We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

This page will explain the most basic CSS selectors.

The CSS element Selector

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

Example

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

```
p {
  text-align: center;
  color: red;
}
```

The CSS id Selector

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

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

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

Example

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

```
#para1 {
  text-align: center;
  color: red;
}
```

Note: An id name cannot start with a number!

The CSS class Selector

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

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

Example

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

```
.center {
  text-align: center;
  color: red;
}
```

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

Example

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

```
p.center {
  text-align: center;
  color: red;
}
```

HTML elements can also refer to more than one class.

Example

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

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

Note: A class name cannot start with a number!

The CSS Universal Selector

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

Example

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

```
* {
  text-align: center;
  color: blue;
}
```

The CSS Grouping Selector

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

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

h2 {
  text-align: center;
  color: red;
}

p {
  text-align: center;
  color: red;
}
```

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

To group selectors, separate each selector with a comma.

Example

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

```
h1, h2, p {
  text-align: center;
  color: red;
}
```

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

All CSS Simple Selectors

Selector	Example	Example description
.class	.intro	Selects all elements with class="intro"
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements
element	p	Selects all <p> elements
element,element,...	div, p	Selects all elements and all <p> elements

How To Add CSS

When a browser reads a style sheet, it will format the HTML document according to the information in the style sheet.

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External CSS
- Internal CSS
- Inline CSS

External CSS

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

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

Example

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

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

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

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

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

"mystyle.css"

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}
```

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;`

Internal CSS

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

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

Example

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

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Inline CSS

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

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

Example

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

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

Tip: An inline style loses many of the advantages of a style sheet (by mixing content with presentation). Use this method sparingly.

Multiple Style Sheets

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

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

```
h1 {
  color: navy;
}
```

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

```
h1 {
  color: orange;
}
```

Example

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

```
<head>
<link rel="stylesheet" type="text/css"
href="mystyle.css">
<style>
h1 {
  color: orange;
}
</style>
</head>
```

Example

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

```
<head>
<style>
h1 {
  color: orange;
}
</style>
<link rel="stylesheet" type="text/css"
href="mystyle.css">
</head>
```

Cascading Order

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

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

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

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

CSS Comments

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers.

A CSS comment starts with /* and ends with */:

Example

```
/* This is a single-line comment */
p {
  color: red;
}
```

You can add comments wherever you want in the code:

Example

```
p {
  color: red;  /* Set text color to red */
}
```

Comments can also span multiple lines:

Example

```
/* This is
a multi-line
comment */

p {
  color: red;
}
```

CSS Colors

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

CSS Color Names

In CSS, a color can be specified by using a color name:

CSS/HTML support 140 standard color names.

CSS Background Color

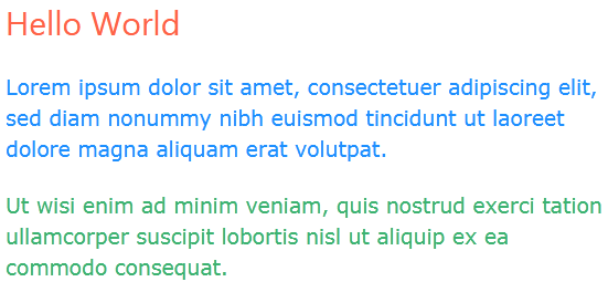
You can set the background color for HTML elements:

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

CSS Text Color

You can set the color of text:



Example

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

CSS Border Color

You can set the color of borders:

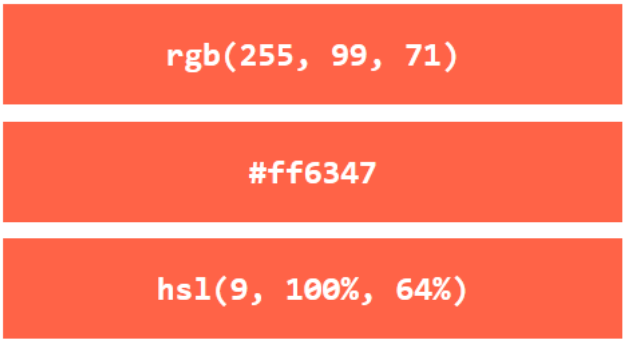
Example

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

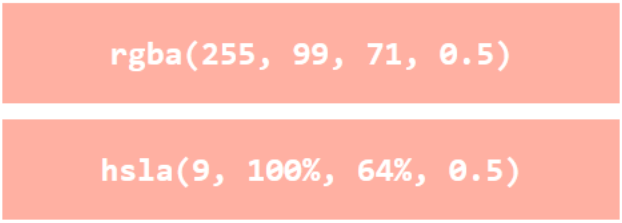
CSS Color Values

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":



Same as color name "Tomato", but 50% transparent:



Example

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71,
0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%,
0.5);">...</h1>
```

CSS RGB Colors

RGB Value

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

rgb(*red*, *green*, *blue*)

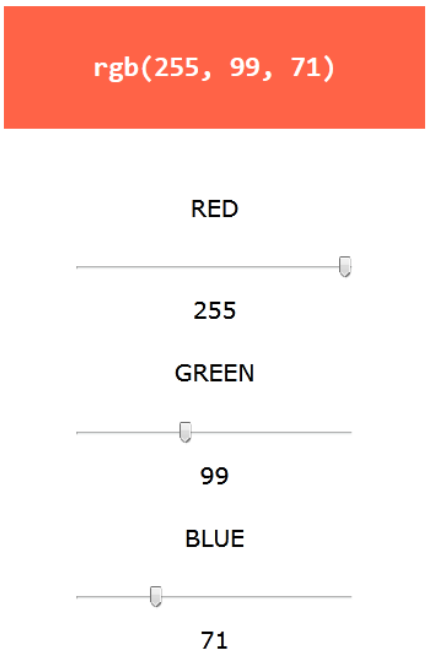
Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.

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

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

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

Experiment by mixing the RGB values below:



Example

rgb(255, 0, 0)

rgb(0, 0, 255)

rgb(60, 179, 113)

rgb(238, 130, 238)

rgb(255, 165, 0)

rgb(106, 90, 205)

Shades of gray are often defined using equal values for all the 3 light sources:

Example

rgb(0, 0, 0)

rgb(60, 60, 60)

rgb(120, 120, 120)

rgb(180, 180, 180)

rgb(240, 240, 240)

rgb(255, 255, 255)

RGBA Value

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.

An RGBA color value is specified with:

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

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

Experiment by mixing the RGBA values below:

rgba(255, 99, 71, 0.5)

RED

255

GREEN

99

BLUE

71

ALPHA

0.5

Example

rgba(255, 99, 71, 0)

rgba(255, 99, 71, 0.2)

rgba(255, 99, 71, 0.4)

rgba(255, 99, 71, 0.6)

rgba(255, 99, 71, 0.8)

rgba(255, 99, 71, 1)

CSS HEX Colors

HEX Value

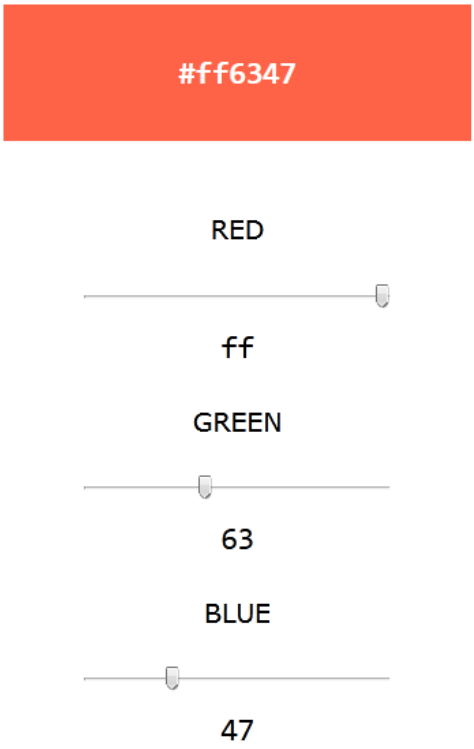
In CSS, a color can be specified using a hexadecimal value in the form:

#rrggbb

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

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

Experiment by mixing the HEX values below:



Example

Shades of gray are often defined using equal values for all the 3 light sources:

Example

#000000

#3c3c3c

#787878

#b4b4b4

#f0f0f0

#ffffff

hsl(0, 100%, 50%)

HUE

0

SATURATION

100%

LIGHTNESS

50%

Example

CSS HSL Colors

HSL Value

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

hsl(*hue*, *saturation*, *lightness*)

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

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

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

Experiment by mixing the HSL values below:

hsl(0, 100%, 50%)

hsl(240, 100%, 50%)

hsl(147, 50%, 47%)

hsl(300, 76%, 72%)

hsl(39, 100%, 50%)

hsl(248, 53%, 58%)

Saturation

Saturation can be described as the intensity of a color.

100% is pure color, no shades of gray

50% is 50% gray, but you can still see the color.

0% is completely gray, you can no longer see the color.

Example

hsl(0, 100%, 50%)

hsl(0, 80%, 50%)

hsl(0, 60%, 50%)

hsl(0, 40%, 50%)

hsl(0, 20%, 50%)

hsl(0, 0%, 50%)

Lightness

The lightness of a color can be described as how much light you want to give the color, where 0% means no light (black), 50% means 50% light (neither dark nor light) 100% means full lightness (white).

Example

hsl(0, 100%, 0%)

hsl(0, 100%, 25%)

hsl(0, 100%, 50%)

hsl(0, 100%, 75%)

hsl(0, 100%, 90%)

hsl(0, 100%, 100%)

Shades of gray are often defined by setting the hue and saturation to 0, and adjust the lightness from 0% to 100% to get darker/lighter shades:

Example

hsl(0, 0%, 0%)

hsl(0, 0%, 24%)

hsl(0, 0%, 47%)

hsl(0, 0%, 71%)

hsl(0, 0%, 94%)

hsl(0, 0%, 100%)

HSLA Value

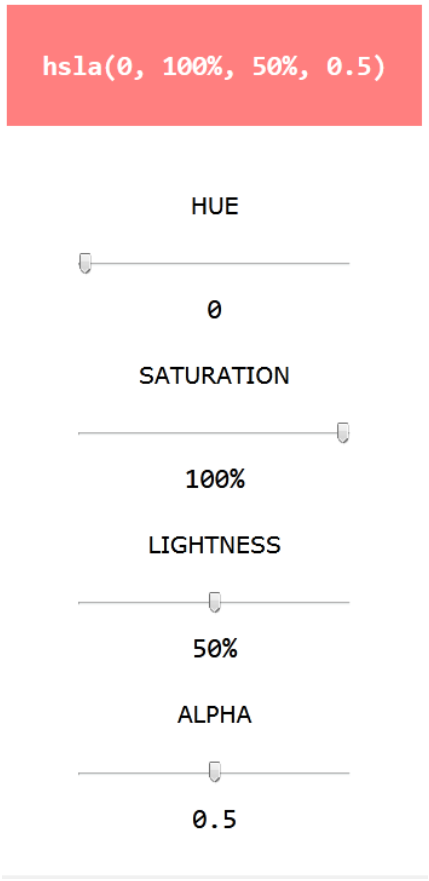
HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.

An HSLA color value is specified with:

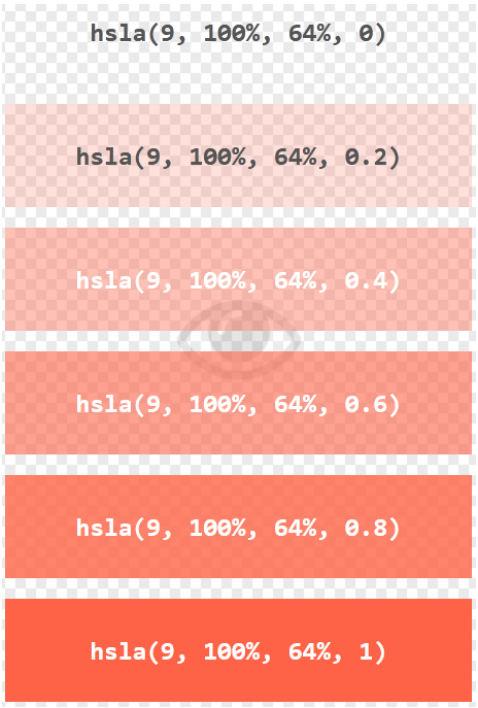
hsla(*hue, saturation, lightness, alpha*)

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

Experiment by mixing the HSLA values below:



Example



CSS 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**

CSS background-color

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

Example

The background color of a page is set like this:

```
body {
  background-color: lightblue;
}
```

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)"

Look at CSS Color Values for a complete list of possible color values.

Example

Here, the <h1>, <p>, and elements will have different background colors:

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

div {
  background-color: lightblue;
}

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

CSS Background Image

CSS background-image

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

Example

The background image for a page can be set like this:

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

Example

This example shows a **bad combination** of text and background image. The text is hardly readable:

```
body {
  background-image: url("bgdesert.jpg");
}
```

Note: When using a background image, use an image that does not disturb the text.

CSS Background Repeat

CSS background-repeat

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

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

Example

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

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

Example

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

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

CSS background-repeat: no-repeat

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

Example

Show the background image only once:

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

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

CSS background-position

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

Example

Position the background image in the top-right corner:

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

CSS Background Attachment

CSS background-attachment

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

Example

Specify that the background image should be fixed:

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

Example

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

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

CSS Background Shorthand

CSS background - Shorthand property

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

The shorthand property for background is **background**.

Example

Use the shorthand property to set all the background properties in one declaration:

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

When using the shorthand property the order of the property values is:

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

It does not matter if one of the property values is missing, as long as the other ones are in this order.

It does not matter if one of the property values is missing, as long as the other ones are in this order.

All CSS Background Properties

Property	Description
background	Sets all the background properties in one declaration
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page
background-clip	Specifies the painting area of the background
background-color	Sets the background color of an element
background-image	Sets the background image for an element
background-origin	Specifies where the background image(s) is/are positioned
background-position	Sets the starting position of a background image
background-repeat	Sets how a background image will be repeated
background-size	Specifies the size of the background image(s)

CSS Borders

CSS Border Properties

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

I have borders on all sides.

I have a red bottom border.

I have rounded borders.

I have a blue left border.

CSS 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).

Example

Demonstration of the different border styles:

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}
p.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
```

Result:

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.

Note: None of the OTHER CSS border properties described below will have ANY effect unless the **border-style** property is set!

CSS Border Width

CSS 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:

Example

Demonstration of the different border widths:

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

p.two {
  border-style: solid;
  border-width: medium;
}

p.three {
  border-style: dotted;
  border-width: 2px;
}

p.four {
  border-style: dotted;
  border-width: thick;
}
```

Result:

5px border-width

medium border-width

2px border-width

thick border-width

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):

Example

```
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 */
}

p.three {
  border-style: solid;
  border-width: 25px 10px 4px 35px; /* 25px
top, 10px right, 4px bottom and 35px left
}
```

CSS Border Color

CSS 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

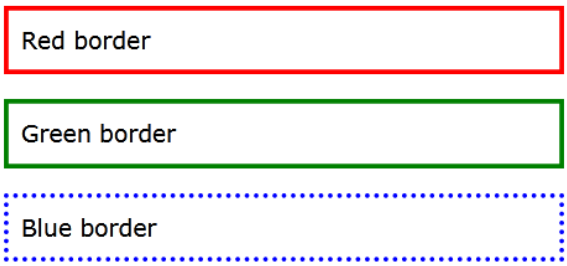
Note: If **border-color** is not set, it inherits the color of the element.

Example

Demonstration of the different border colors:

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

p.two {
  border-style: solid;
  border-color: green;
}

p.three {
  border-style: dotted;
  border-color: blue;
}
```

Result:



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).

Example

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

HEX Values

The color of the border can also be specified using a hexadecimal value (HEX):

Example

```
p.one {
  border-style: solid;
  border-color: #ff0000; /* red */
}
```

RGB Values

Or by using RGB values:

Example

```
p.one {
  border-style: solid;
  border-color: rgb(255, 0, 0); /* red */
}
```

HSL Values

You can also use HSL values:

Example

```
p.one {
  border-style: solid;
  border-color: hsl(0, 100%, 50%); /* red */
}
```

CSS Border Sides

CSS 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):

Example

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

Result:

Different Border Styles

The example above gives the same result as this:

Example

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

So, here is how it works:

If the **border-style** property has four values:

- **border-style: dotted solid double dashed;**
  - top border is dotted
  - right border is solid
  - bottom border is double
  - left border is dashed

If the **border-style** property has three values:

- **border-style: dotted solid double;**
  - top border is dotted
  - right and left borders are solid
  - bottom border is double

If the **border-style** property has two values:

- **border-style: dotted solid;**
  - top and bottom borders are dotted
  - right and left borders are solid

If the **border-style** property has one value:

- **border-style: dotted;**
  - all four borders are dotted

Example

```
/* Four values */
p {
  border-style: dotted solid double dashed;
}

/* Three values */
p {
  border-style: dotted solid double;
}

/* Two values */
p {
  border-style: dotted solid;
}

/* One value */
p {
  border-style: dotted;
}
```

The **border-style** property is used in the example above. However, it also works with **border-width** and **border-color**.

CSS Shorthand Border Property

Like you saw in the previous page, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the individual border properties in one property.

The **border** property is a shorthand property for the following individual border properties:

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

Example

```
p {
  border: 5px solid red;
}
```

Result:

Some text

You can also specify all the individual border properties for just one side:

Left Border

```
p {
  border-left: 6px solid red;
  background-color: lightgrey;
}
```

Result:

Some text

Bottom Border

```
p {
  border-bottom: 6px solid red;
  background-color: lightgrey;
}
```

Result:

Some text

CSS Rounded Borders

CSS Rounded Borders

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

Normal border

Round border

Rounder border

Roundest border

Example

```
p {
  border: 2px solid red;
  border-radius: 5px;
}
```

More Examples

All the top border properties in one declaration
This example demonstrates a shorthand property for setting all of the properties for the top border in one declaration.

Set the style of the bottom border
This example demonstrates how to set the style of the bottom border.

Set the width of the left border
This example demonstrates how to set the width of the left border.

Set the color of the four borders
This example demonstrates how to set the color of the four borders. It can have from one to four colors.

Set the color of the right border
This example demonstrates how to set the color of the right border.

All CSS Border Properties

Property	Description
border	Sets all the border properties in one declaration
border-bottom	Sets all the bottom border properties in one declaration
border-bottom-color	Sets the color of the bottom border
border-bottom-style	Sets the style of the bottom border
border-bottom-width	Sets the width of the bottom border
border-color	Sets the color of the four borders
border-left	Sets all the left border properties in one declaration
border-left-color	Sets the color of the left border
border-left-style	Sets the style of the left border
border-left-width	Sets the width of the left border
border-radius	Sets all the four border-*-radius properties for rounded corners
border-right	Sets all the right border properties in one declaration
border-right-color	Sets the color of the right border
border-right-style	Sets the style of the right border
border-right-width	Sets the width of the right border

border-style	Sets the style of the four borders
border-top	Sets all the top border properties in one declaration
border-top-color	Sets the color of the top border
border-top-style	Sets the style of the top border
border-top-width	Sets the width of the top border
border-width	Sets the width of the four borders

CSS Margins

This element has a margin of 70px.

CSS Margins

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

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

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

Tip: Negative values are allowed.

Example

Set different margins for all four sides of a <p> element:

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

Margin - Shorthand Property

To shorten the code, it is possible to specify all the margin properties in one property.

The **margin** property is a shorthand property for the following individual margin properties:

- **margin-top**
- **margin-right**
- **margin-bottom**
- **margin-left**

So, here is how it works:

If the **margin** property has four values:

- **margin: 25px 50px 75px 100px;**
  - top margin is 25px
  - right margin is 50px
  - bottom margin is 75px
  - left margin is 100px

Example

Use the margin shorthand property with four values:

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

If the **margin** property has three values:

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

Example

Use the margin shorthand property with three values:

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

If the margin property has two values:

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

Example

Use the margin shorthand property with two values:

```
p {
  margin: 25px 50px;
}
```

If the margin property has one value:

- **margin: 25px;**
  - all four margins are 25px

Example

Use the margin shorthand property with one value:

```
p {
  margin: 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.

Example

Use margin: auto:

```
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 ( ):

Example

Use of the inherit value:

```
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!

Look at the following example:

Example

Demonstration of margin collapse:

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

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

In the example above, the <h1> element has a bottom margin of 50px and the <h2> element has a top margin set to 20px.

Common sense would seem to suggest that the vertical margin between the <h1> and the <h2> would be a total of 70px (50px + 20px). But due to margin collapse, the actual margin ends up being 50px.

All CSS Margin Properties

Property	Description
margin	A shorthand property for setting the margin properties in one declaration
margin-bottom	Sets the bottom margin of an element
margin-left	Sets the left margin of an element
margin-right	Sets the right margin of an element
margin-top	Sets the top margin of an element

CSS Padding

CSS 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.

Example

Set different padding for all four sides of a element:

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

Padding - Shorthand Property

To shorten the code, it is possible to specify all the padding properties in one property.

The **padding** property is a shorthand property for the following individual padding properties:

- **padding-top**
- **padding-right**
- **padding-bottom**
- **padding-left**

So, here is how it works:

If the **padding** property has four values:

- **padding: 25px 50px 75px 100px;**
  - top padding is 25px
  - right padding is 50px
  - bottom padding is 75px
  - left padding is 100px

Example

Use the padding shorthand property with four values:

```
div {
  padding: 25px 50px 75px 100px;
}
```

If the **padding** property has three values:

- **padding: 25px 50px 75px;**
  - top padding is 25px
  - right and left paddings are 50px
  - bottom padding is 75px

Example

Use the padding shorthand property with three values:

```
div {
  padding: 25px 50px 75px;
}
```

If the **padding** property has two values:

- **padding: 25px 50px;**
  - top and bottom paddings are 25px
  - right and left paddings are 50px

Example

Use the padding shorthand property with two values:

```
div {
  padding: 25px 50px;
}
```

If the **padding** property has one value:

- **padding: 25px;**
  - all four paddings are 25px

Example

Use the padding shorthand property with one value:

```
div {
  padding: 25px;
}
```

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.

Example

Here, the element is given a width of 300px. However, the actual width of the element will be 350px (300px + 25px of left padding + 25px of right padding):

```
div {
  width: 300px;
  padding: 25px;
}
```

To keep the width at 300px, no matter the amount of padding, you can use the box-sizing property. This causes the element to maintain its width; if you increase the padding, the available content space will decrease.

Example

Use the box-sizing property to keep the width at 300px, no matter the amount of padding:

```
div {
  width: 300px;
  padding: 25px;
  box-sizing: border-box;
}
```

More Examples

Set the left padding
This example demonstrates how to set the left padding of a <p> element.

Set the right padding
This example demonstrates how to set the right padding of a <p> element.

Set the top padding
This example demonstrates how to set the top padding of a <p> element.

Set the bottom padding
This example demonstrates how to set the bottom padding of a <p> element.

All CSS Padding Properties

Property	Description
padding	A shorthand property for setting all the padding properties in one declaration
padding-bottom	Sets the bottom padding of an element
padding-left	Sets the left padding of an element
padding-right	Sets the right padding of an element

padding-top	Sets the top padding of an element

CSS Height and Width

This element has a width of 100%.

CSS 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/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

CSS height/width Examples

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

Example

Set the height and width of a element:

```
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
```

This element has a height of 100 pixels and a width of 500 pixels.

Example

Set the height and width of another element:

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

Note: Remember that the height and width properties do not include padding, borders, or margins! They set the height/width of the area inside the padding, border, and margin of the element!

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 ** ** 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.

Tip: Drag the browser window to smaller than 500px wide, to see the difference between the two divs!

This element has a height of 100 pixels and a max-width of 500 pixels.

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

Example

This element has a height of 100 pixels and a max-width of 500 pixels:

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

Try it Yourself - Examples

Set the height and width of elements
This example demonstrates how to set the height and width of different elements.

Set the height and width of an image using percent
This example demonstrates how to set the height and width of an image using a percent value.

Set min-width and max-width of an element
This example demonstrates how to set a minimum width and a maximum width of an element using a pixel value.

Set min-height and max-height of an element
This example demonstrates how to set a minimum height and a maximum height of an element using a pixel value.

All CSS Dimension Properties

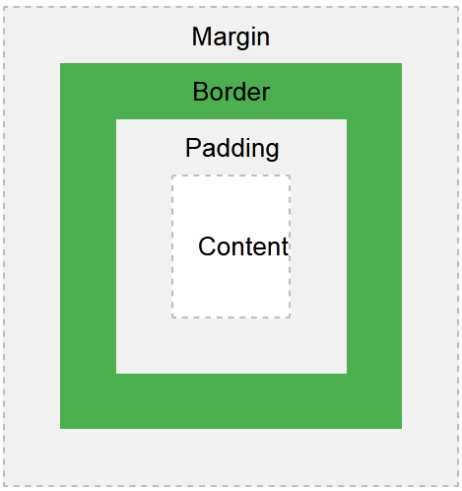
Property	Description
height	Sets the height of an element
max-height	Sets the maximum height of an element
max-width	Sets the maximum width of an element
min-height	Sets the minimum height of an element
min-width	Sets the minimum width of an element
width	Sets the width of an element

CSS Box Model

The CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

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

Example

Demonstration of the box model:

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

Width and Height of an Element

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

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

Example

This element will have a total width of 350px:

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

Here is the calculation:

320px (width)
+ 20px (left + right padding)
+ 10px (left + right border)
+ 0px (left + right margin)
= 350px

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

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

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

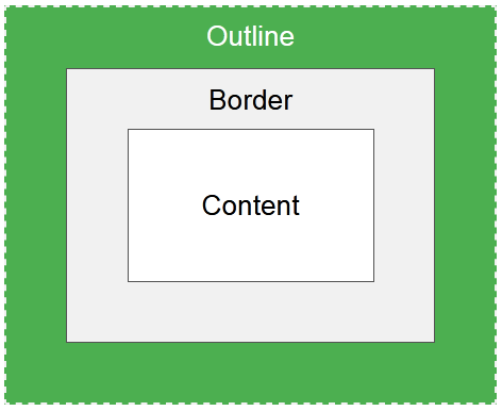
Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

CSS Outline

This element has a black border and a green outline with a width of 10px.

CSS 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

The following example shows the different **outline-style** values:

A dotted outline.

A dashed outline.

A solid outline.

A double outline.

A groove outline. The effect depends on the outline-color value.

A ridge outline. The effect depends on the outline-color value.

An inset outline. The effect depends on the outline-color value.

An outset outline. The effect depends on the outline-color value.

Example

Demonstration of the different outline styles:

```
p.dotted {outline-style: dotted;}
p.dashed {outline-style: dashed;}
p.solid {outline-style: solid;}
p.double {outline-style: double;}
p.groove {outline-style: groove;}
p.ridge {outline-style: ridge;}
p.inset {outline-style: inset;}
p.outset {outline-style: outset;}
```

Note: None of the other outline properties will have any effect, unless the outline-style property is set!

CSS 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"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"
- invert - performs a color inversion (which ensures that the outline is visible, regardless of color background)

The following example shows some different outlines with different colors. Also notice that these elements also have a thin black border inside the outline:

A solid red outline.

A double green outline.

An outset yellow outline.

Example

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

p.ex2 {
  border: 1px solid black;
  outline-style: double;
  outline-color: green;
}

p.ex3 {
  border: 1px solid black;
  outline-style: outset;
  outline-color: yellow;
}
```

The following example uses outline-color: invert, which performs a color inversion. This ensures that the outline is visible, regardless of color background:

A solid invert outline.

Example

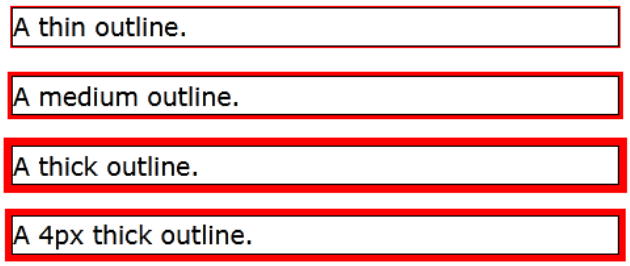
```
p.ex1 {
  border: 1px solid yellow;
  outline-style: solid;
  outline-color: invert;
}
```

CSS Outline Width

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

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

The following example shows some outlines with different widths:



Example

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

p.ex2 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
  outline-width: medium;
}

p.ex3 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
  outline-width: thick;
}

p.ex4 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
  outline-width: 4px;
}
```

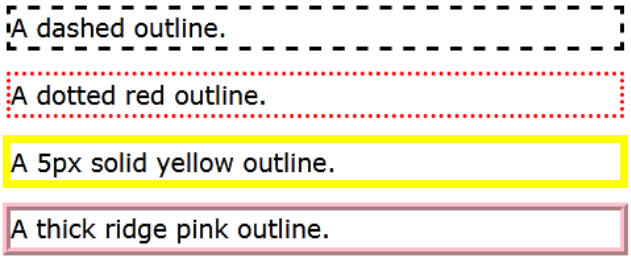
CSS Outline - Shorthand property

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

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

The **outline** property is specified as one, two, or three values from the list above. The order of the values does not matter.

The following example shows some outlines specified with the shorthand **outline** property:



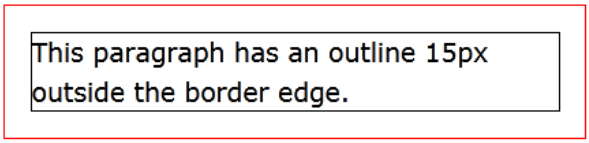
Example

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

CSS Outline Offset

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

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



Example

```
p {
  margin: 30px;
  border: 1px solid black;
  outline: 1px solid red;
  outline-offset: 15px;
}
```

The following example shows that the space between an element and its outline is transparent:

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

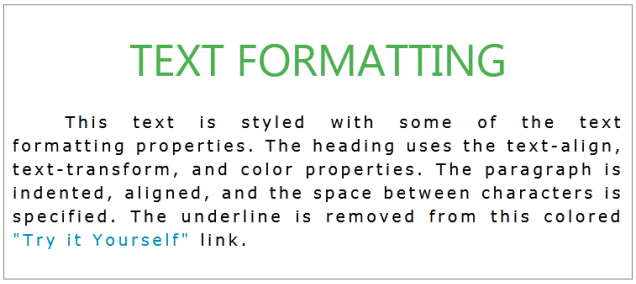
Example

```
p {
  margin: 30px;
  background: yellow;
  border: 1px solid black;
  outline: 1px solid red;
  outline-offset: 15px;
}
```

All CSS Outline Properties

Property	Description
outline	A shorthand property for setting outline-width, outline-style, and outline-color in one declaration
outline-color	Sets the color of an outline
outline-offset	Specifies the space between an outline and the edge or border of an element
outline-style	Sets the style of an outline
outline-width	Sets the width of an outline

CSS 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)"

Look at CSS Color Values for a complete list of possible color values.

The default text color for a page is defined in the body selector.

Example

```
body {
  color: blue;
}

h1 {
  color: green;
}
```

Note: For W3C compliant CSS: If you define the **color** property, you must also define the **background-color**.

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.

The following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left):

Example

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

h2 {
  text-align: left;
}

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

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

Example

```
div {
  text-align: justify;
}
```

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:

Example

```
a {
  text-decoration: none;
}
```

The other **text-decoration** values are used to decorate text:

Example

```
h1 {
  text-decoration: overline;
}

h2 {
  text-decoration: line-through;
}

h3 {
  text-decoration: underline;
}
```

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:

Example

```
p.uppercase {
  text-transform: uppercase;
}

p.lowercase {
  text-transform: lowercase;
}

p.capitalize {
  text-transform: capitalize;
}
```

Text Indentation

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

Example

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

Letter Spacing

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

The following example demonstrates how to increase or decrease the space between characters:

Example

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

h2 {
  letter-spacing: -3px;
}
```

Line Height

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

Example

```
p.small {
  line-height: 0.8;
}

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

Text Direction

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

Example

```
p {
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

Word Spacing

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

The following example demonstrates how to increase or decrease the space between words:

Example

```
h1 {
  word-spacing: 10px;
}

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

Text Shadow

The **text-shadow** property adds shadow to text.

The following example specifies the position of the horizontal shadow (3px), the position of the vertical shadow (2px) and the color of the shadow (red):

Example

```
h1 {
  text-shadow: 3px 2px red;
}
```

More Examples

Disable text wrapping inside an element
This example demonstrates how to disable text wrapping inside an element.

Vertical alignment of an image
This example demonstrates how to set the vertical align of an image in a text.

All CSS Text Properties

Property	Description
color	Sets the color of text
direction	Specifies the text direction/writing direction
letter-spacing	Increases or decreases the space between characters in a text
line-height	Sets the line height
text-align	Specifies the horizontal alignment of text
text-decoration	Specifies the decoration added to text
text-indent	Specifies the indentation of the first line in a text-block
text-shadow	Specifies the shadow effect added to

	text
text-transform	Controls the capitalization of text
text-overflow	Specifies how overflowed content that is not displayed should be signaled to the user
unicode-bidi	Used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document
vertical-align	Sets the vertical alignment of an element
white-space	Specifies how white-space inside an element is handled
word-spacing	Increases or decreases the space between words in a text

CSS Fonts

The CSS font properties define the font family, boldness, size, and the style of a text.

Difference Between Serif and Sans-serif Fonts

CSS Font Families

In CSS, there are two types of font family names:

- **generic family** - a group of font families with a similar look (like "Serif" or "Monospace")
- **font family** - a specific font family (like "Times New Roman" or "Arial")

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters
Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters
Monospace	Courier New Lucida Console	All monospace characters have the same width

Note: On computer screens, sans-serif fonts are considered easier to read than serif 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:

Example

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

For commonly used font combinations, look at our Web Safe Font Combinations.

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)

Example

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

p.italic {
  font-style: italic;
}

p.oblique {
  font-style: oblique;
}
```

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:

Example

```
h1 {
  font-size: 40px;
}

h2 {
  font-size: 30px;
}

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

Tip: If you use pixels, you can still use the zoom tool to resize the entire page.

Set Font Size With Em

To allow users to resize the text (in the browser menu), many developers use em instead of pixels.

The em size unit is recommended by the W3C.

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

The size can be calculated from pixels to em using this formula: *pixels*/16=*em*

Example

```
h1 {
  font-size: 2.5em; /* 40px/16=2.5em */
}

h2 {
  font-size: 1.875em; /* 30px/16=1.875em */
}

p {
  font-size: 0.875em; /* 14px/16=0.875em */
}
```

In the example above, the text size in em is the same as the previous example in pixels. However, with the em size, it is possible to adjust the text size in all browsers.

Unfortunately, there is still a problem with older versions of IE. The text becomes larger than it should when made larger, and smaller than it should when made smaller.

Use a Combination of Percent and Em

The solution that works in all browsers, is to set a default font-size in percent for the <body> element:

Example

```
body {
  font-size: 100%;
}

h1 {
  font-size: 2.5em;
}

h2 {
  font-size: 1.875em;
}

p {
  font-size: 0.875em;
}
```

Our code now works great! It shows the same text size in all browsers, and allows all browsers to zoom or resize the text!

Font Weight

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

Example

```
p.normal {
  font-weight: normal;
}

p.thick {
  font-weight: bold;
}
```

Responsive Font Size

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

That way the text size will follow the size of the browser window:

Example

```
<h1 style="font-size:10vw">Hello World</h1>
```

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

Font Variant

The **font-variant** property specifies whether or not a text should be displayed in a small-caps font.

In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.

Example

```
p.normal {
  font-variant: normal;
}

p.small {
  font-variant: small-caps;
}
```

All CSS Font Properties

Property	Description
font	Sets all the font properties in one declaration
font-family	Specifies the font family for text
font-size	Specifies the font size of text
font-style	Specifies the font style for text
font-variant	Specifies whether or not a text should be displayed in a small-caps font
font-weight	Specifies the weight of a font

CSS Icons

How To Add Icons

The simplest way to add an icon to your HTML page, is with an icon library, such as Font Awesome.

Add the name of the specified icon class to any inline HTML element (like **<i>** or ****).

All the icons in the icon libraries below, are scalable vectors that can be customized with CSS (size, color, shadow, etc.)

Font Awesome Icons

To use the Font Awesome icons, go to fontawesome.com, sign in, and get a code to add in the **<head>** section of your HTML page:

<script src="https://kit.fontawesome.com/*yourcode*.js"></script>

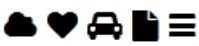
Read more about how to get started with Font Awesome in our Font Awesome 5 tutorial.

Note: No downloading or installation is required!

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://kit.fontawesome.com
/a076d05399.js"></script>
</head>
<body>

<i class="fas fa-cloud"></i>
<i class="fas fa-heart"></i>
<i class="fas fa-car"></i>
<i class="fas fa-file"></i>
<i class="fas fa-bars"></i>

</body>
</html>
```

Result:



Bootstrap Icons

To use the Bootstrap glyphicons, add the following line inside the **<head>** section of your HTML page:

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

Note: No downloading or installation is required!

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com
/bootstrap/3.3.7/css/bootstrap.min.css">
</head>
<body>

<i class="glyphicon glyphicon-cloud"></i>
<i class="glyphicon glyphicon-remove"></i>
<i class="glyphicon glyphicon-user"></i>
<i class="glyphicon glyphicon-envelope"></i>
<i class="glyphicon glyphicon-thumbs-
up"></i>

</body>
</html>
```

Result:

Google Icons

To use the Google icons, add the following line inside the **<head>** section of your HTML page:

<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+
Icons">

Note: No downloading or installation is required!

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet"
href="https://fonts.googleapis.com
/icon?family=Material+Icons">
</head>
<body>

<i class="material-icons">cloud</i>
<i class="material-icons">favorite</i>
<i class="material-icons">attachment</i>
<i class="material-icons">computer</i>
<i class="material-icons">traffic</i>

</body>
</html>
```

Result:

CSS Links

With CSS, links can be styled in different ways.

Styling Links

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

Example

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

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

Example

```
/* unvisited link */
a:link {
  color: red;
}

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

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

/* selected link */
a:active {
  color: blue;
}
```

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

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

Text Decoration

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

Example

```
a:link {
  text-decoration: none;
}

a:visited {
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

a:active {
  text-decoration: underline;
}
```

Background Color

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

Example

```
a:link {
  background-color: yellow;
}

a:visited {
  background-color: cyan;
}

a:hover {
  background-color: lightgreen;
}

a:active {
  background-color: hotpink;
}
```

Link Buttons

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

Example

```
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;
}
```

More Examples

Add different styles to hyperlinks
This example demonstrates how to add other styles to hyperlinks.

Advanced - Create a link button with borders
Another example of how to create link boxes/buttons.

Change the cursor
The cursor property specifies the type of cursor to display.
This example demonstrates the different types of cursors (can be useful for links).

CSS 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.

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

Example

```
ul.a {
  list-style-type: circle;
}

ul.b {
  list-style-type: square;
}

ol.c {
  list-style-type: upper-roman;
}

ol.d {
  list-style-type: lower-alpha;
}
```

Note: Some of the values are for unordered lists, and some for ordered lists.

An Image as The List Item Marker

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

Example

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

Position The List Item Markers

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

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

- 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

Example

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

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

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>**:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

List - Shorthand property

The **list-style** property is a shorthand property. It is used to set all the list properties in one declaration:

Example

```
ul {
  list-style: square inside
url("sqpurple.gif");
}
```

When using the shorthand property, the order of the property values are:

- **list-style-type** (if a list-style-image is specified, the value of this property will be displayed if the image for some reason cannot be displayed)
- **list-style-position** (specifies whether the list-item markers should appear inside or outside the content flow)
- **list-style-image** (specifies an image as the list item marker)

If one of the property values above are missing, the default value for the missing property will be inserted, if any.

Styling List With Colors

We can also style lists with colors, to make them look a little more interesting.

Anything added to the **<ol>** or **<ul>** tag, affects the entire list, while properties added to the **<li>** tag will affect the individual list items:

Example

```
ol {
  background: #ff9999;
  padding: 20px;
}

ul {
  background: #3399ff;
  padding: 20px;
}

ol li {
  background: #ffe5e5;
  padding: 5px;
  margin-left: 35px;
}

ul li {
  background: #cce5ff;
  margin: 5px;
}
```

Result:

More Examples

Customized list with a red left border
This example demonstrates how to create a list with a red left border.

Full-width bordered list
This example demonstrates how to create a bordered list without bullets.

All the different list-item markers for lists
This example demonstrates all the different list-item markers in CSS.

All CSS List Properties

Property	Description
list-style	Sets all the properties for a list in one declaration
list-style-image	Specifies an image as the list-item marker
list-style-position	Specifies the position of the list-item markers (bullet points)
list-style-type	Specifies the type of list-item marker

CSS Tables

The look of an HTML table can be greatly improved with CSS:

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Berglunds snabbköp	Christina Berglund	Sweden
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Königlich Essen	Philip Cramer	Germany
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

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;
}
```

Notice that the table in the example above has double borders. This is because both the table and the **<th>** and **<td>** elements have separate borders.

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;
}
```

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;
}
```

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;
}
```

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;
}
```

Vertical Alignment

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;
}
```

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

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;
}
```

Hoverable Table

Use the **:hover** selector on **<tr>** to highlight table rows on mouse over:

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

Example

```
tr:hover {background-color: #f5f5f5;}
```

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;}
```

Table Color

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

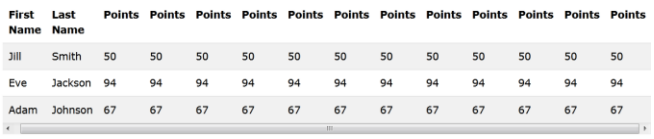
First Name	Last Name	Savings
Peter	Griffin	$100
Lois	Griffin	$150
Joe	Swanson	$300

Example

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

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	Points	Points
Jill	Smith	50	50	50	50	50	50	50	50	50	50	50	50
Eve	Jackson	94	94	94	94	94	94	94	94	94	94	94	94
Adam	Johnson	67	67	67	67	67	67	67	67	67	67	67	67

Add a container element (like ** **) 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).

More Examples

Make a fancy table
This example demonstrates how to create a fancy table.

Set the position of the table caption
This example demonstrates how to position the table caption.

CSS Table Properties

Property	Description
border	Sets all the border properties in one declaration
border-collapse	Specifies whether or not table borders should be collapsed
border-spacing	Specifies the distance between the borders of adjacent cells
caption-side	Specifies the placement of a table caption
empty-cells	Specifies whether or not to display borders and background on empty cells in a table
table-layout	Sets the layout algorithm to be used for a table

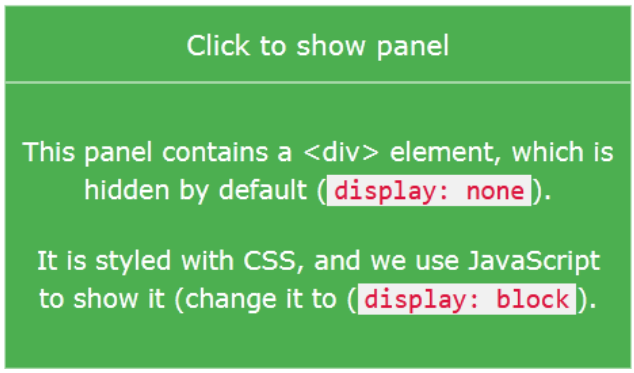
CSS Layout - The display Property

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

The display Property

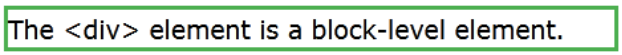
The **display** property specifies if/how an element is displayed.

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



Block-level Elements

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

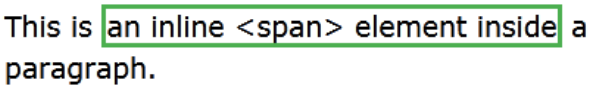


Examples of block-level elements:

-
- <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.



Examples of inline elements:

-
- <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:

Example

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

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

The following example displays elements as block elements:

Example

```
span {
  display: block;
}
```

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

Example

```
a {
  display: block;
}
```

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

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

Example

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

visibility:hidden; also hides an element.

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

Example

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

More Examples

Differences between display: none; and visibility: hidden;
This example demonstrates display: none; versus visibility: hidden;

Using CSS together with JavaScript to show content
This example demonstrates how to use CSS and JavaScript to show an element on click.

CSS Display/Visibility Properties

Property	Description
display	Specifies how an element should be displayed
visibility	Specifies whether or not an element should be visible

CSS 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 element has a width of 500px, and margin set to auto.

Note: The problem with the ** ** 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 element has a max-width of 500px, and margin set to auto.

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

Here is an example of the two divs above:

Example

```
div.ex1 {
  width: 500px;
  margin: auto;
  border: 3px solid #73AD21;
}

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

CSS Layout - The position Property

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

The position Property

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

There are five different position values:

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

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

position: static;

HTML elements are positioned static by default.

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

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

This element has position: static;

Here is the CSS that is used:

Example

```
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 element has position: relative;

Here is the CSS that is used:

Example

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

position: fixed;

An element with **position: fixed;** is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

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

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

Example

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

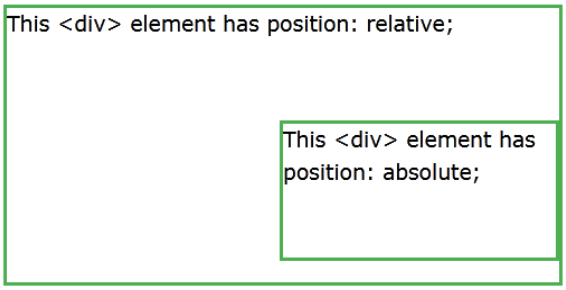
position: absolute;

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

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

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

Here is a simple example:



Here is the CSS that is used:

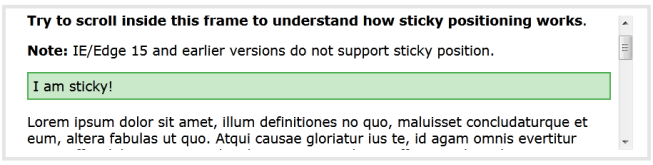
Example

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

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

position: sticky;

An element with **position: sticky;** is positioned based on the user's scroll position.

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



Note: Internet Explorer, Edge 15 and earlier versions do not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of **top**, **right**, **bottom** or **left** for sticky positioning to work.

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

Example

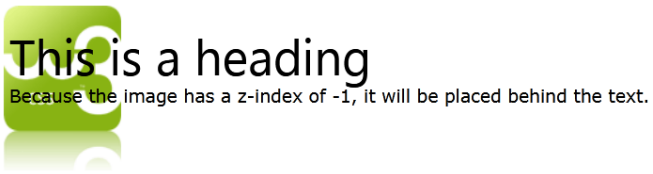
```
div.sticky {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
  background-color: green;
  border: 2px solid #4CAF50;
}
```

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:



Example

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

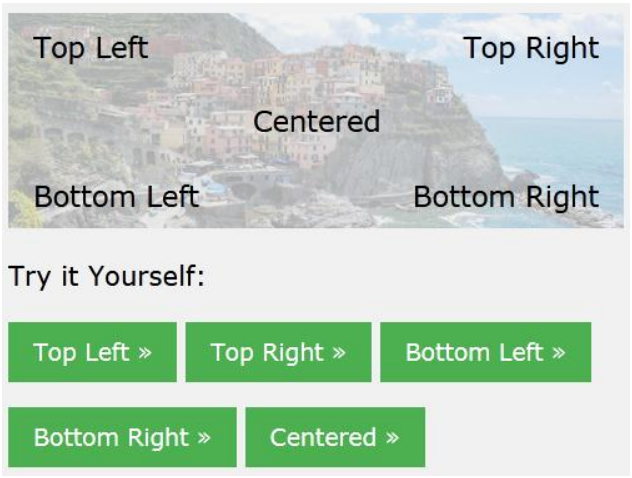
An element with greater stack order is always in front of an element with a lower stack order.

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

Positioning Text In an Image

How to position text over an image:

Example



More Examples

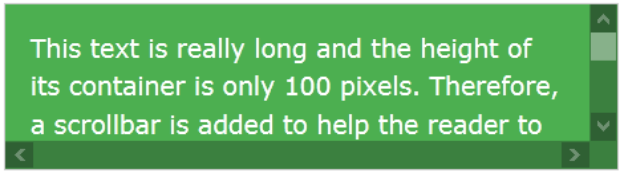
Set the shape of an element
This example demonstrates how to set the shape of an element. The element is clipped into this shape, and displayed.

All CSS Positioning Properties

Property	Description
bottom	Sets the bottom margin edge for a positioned box
clip	Clips an absolutely positioned element
left	Sets the left margin edge for a positioned box
position	Specifies the type of positioning for an element
right	Sets the right margin edge for a positioned box
top	Sets the top margin edge for a positioned box
z-index	Sets the stack order of an element

CSS Layout - Overflow

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



CSS Overflow

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

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

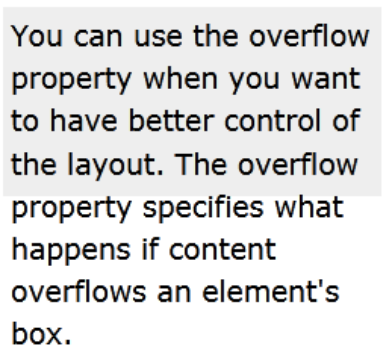
- **visible** - Default. The overflow is not clipped. The content renders outside the element's box
- **hidden** - The overflow is clipped, and the rest of the content will be invisible
- **scroll** - The overflow is clipped, and a scrollbar is added to see the rest of the content
- **auto** - Similar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

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

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

overflow: visible

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

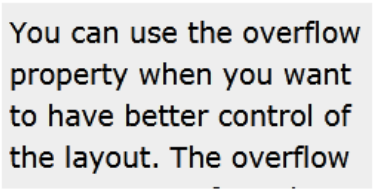


Example

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

overflow: hidden

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

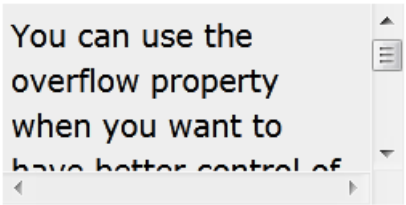


Example

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

overflow: scroll

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

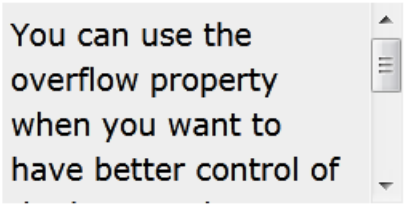


Example

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

overflow: auto

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



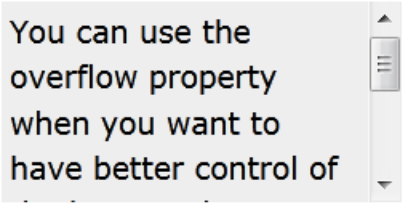
Example

```
div {
  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.



Example

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

All CSS Overflow Properties

Property	Description
overflow	Specifies what happens if content overflows an element's box
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

CSS Layout - float and clear

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

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

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.

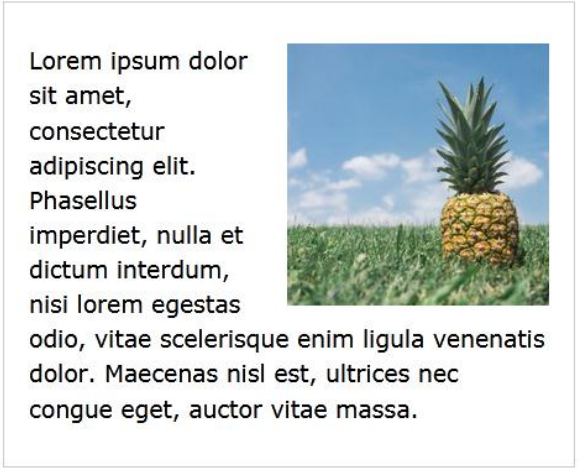
The **float** property can have one of the following values:

- 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:



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.

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.

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.

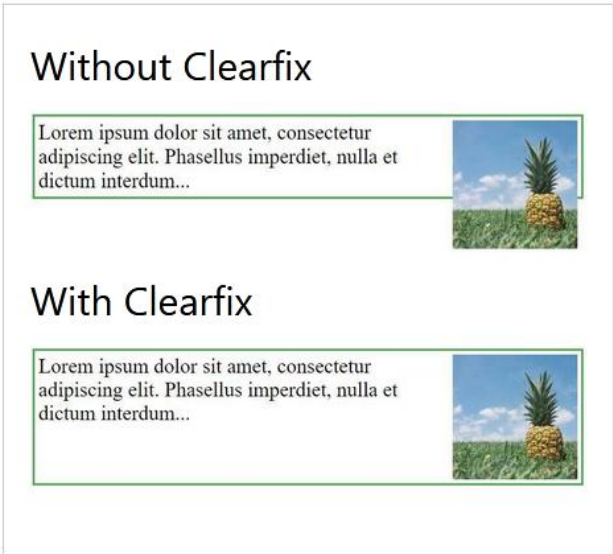
The following example clears the float to the left. Means that no floating elements are allowed on the left side (of the div):

Example

```
div {
  clear: left;
}
```

The clearfix Hack

If an element is taller than the element containing it, and it is floated, it will "overflow" outside of its container:



Then we can add **overflow: auto;** to the containing element to fix this problem:

Example

```
.clearfix {
  overflow: auto;
}
```

The **overflow: auto** clearfix works well as long as you are able to keep control of your margins and padding (else you might see scrollbars). The **new, modern clearfix hack** however, is safer to use, and the following code is used for most webpages:

Example

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

You will learn more about the **::after** pseudo-element in a later chapter.

Grid of Boxes / Equal Width Boxes

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

Example

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

.box {
  float: left;
  width: 33.33%; /* three boxes (use 25% for four, and 50% for two, etc) */
  padding: 50px; /* if you want space between the images */
}
```

What is box-sizing?

You can easily create three floating boxes side by side. However, when you add something that enlarges the width of each box (e.g. padding or borders), the box will break. The **box-sizing** property allows us to include the padding and border in the box's total width (and height), making sure that the padding stays inside of the box and that it does not break.

You can read more about the box-sizing property in our CSS Box Sizing Chapter.

Images Side By Side

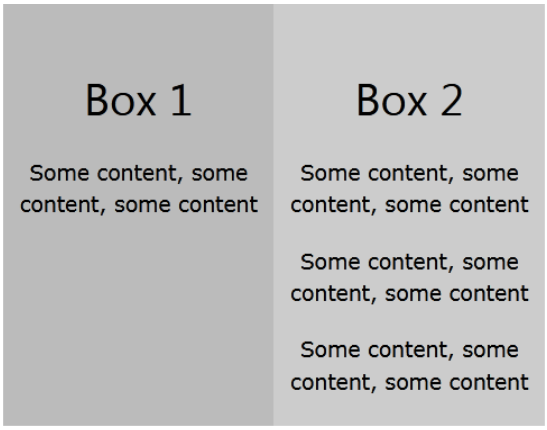
The grid of boxes can also be used to display images side by side:

Example

```
.img-container {
  float: left;
  width: 33.33%; /* three containers (use 25% for four, and 50% for two, etc) */
  padding: 5px; /* if you want space between the images */
}
```

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:



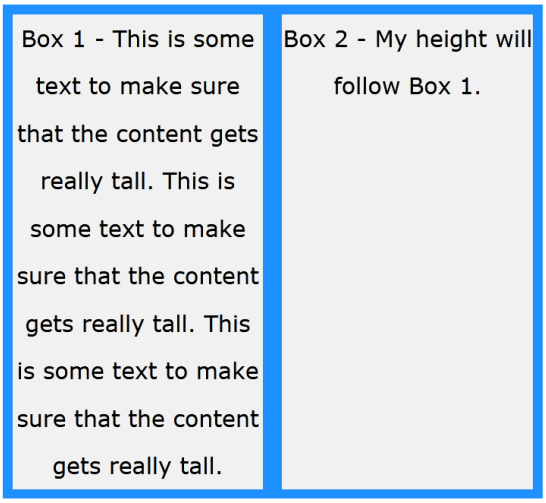
Example

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

However, this is not very flexible. It is ok if you can guarantee that the boxes will always have the same amount of content in them. But many times, the content is not the same. If you try the example above on a mobile phone, you will see that the second box's content will be displayed outside of the box. This is where CSS3 Flexbox comes in handy - as it can automatically stretch boxes to be as long as the longest box:

Example

Using **Flexbox** to create flexible boxes:

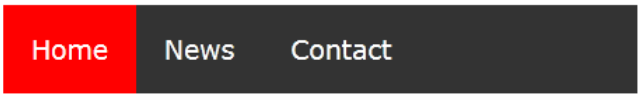


The only problem with Flexbox is that it does not work in Internet Explorer 10 or earlier versions. You can read more about the Flexbox Layout Module in our CSS Flexbox Chapter.

Navigation Menu

Use **float** with a list of hyperlinks to create a horizontal menu:

Example



Web Layout Example

It is also common to do entire web layouts using the **float** property:

Example

```
.header, .footer {
  background-color: grey;
  color: white;
  padding: 15px;
}

.column {
  float: left;
  padding: 15px;
}

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

.menu {
  width: 25%;
}

.content {
  width: 75%;
}
```

More Examples

An image with border and margins that floats to the right in a paragraph
Let an image float to the right in a paragraph. Add border and margins to the image.

An image with a caption that floats to the right
Let an image with a caption float to the right.

Let the first letter of a paragraph float to the left
Let the first letter of a paragraph float to the left and style the letter.

Creating a website with float
Use float to create a homepage with a navbar, header, footer, left content and main content.

All CSS Float Properties

Property	Description
box-sizing	Defines how the width and height of an element are calculated: should they include padding and borders, or not
clear	Specifies what elements can float beside the cleared element and on which side
float	Specifies how an element should float
overflow	Specifies what happens if content overflows an element's box
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

CSS Layout - display: inline-block

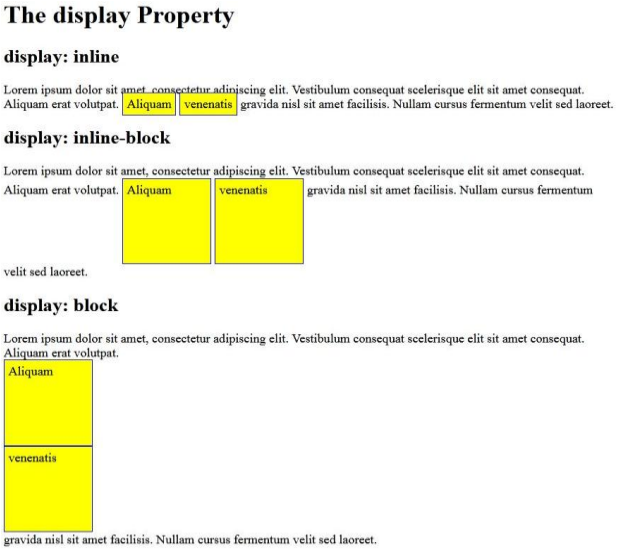
The display: inline-block Value

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

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

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

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



Example

```
span.a {
  display: inline; /* the default for span
*/
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}

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

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

Using inline-block to Create Navigation Links

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

Example

```
.nav {
  background-color: yellow;
  list-style-type: none;
  text-align: center;
  padding: 0;
  margin: 0;
}

.nav li {
  display: inline-block;
  font-size: 20px;
  padding: 20px;
}
```

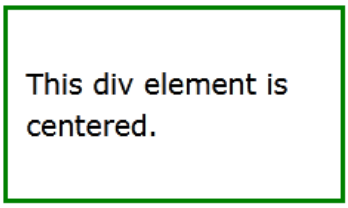
CSS Layout - Horizontal & Vertical Align

Center Align Elements

To horizontally center a block element (like), 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:



Example

```
.center {
  margin: auto;
  width: 50%;
  border: 3px solid green;
  padding: 10px;
}
```

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

Center Align Text

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

This text is centered.

Example

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

Center an Image

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

Example

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

Left and Right Align - Using position

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

Example

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

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

Left and Right Align - Using float

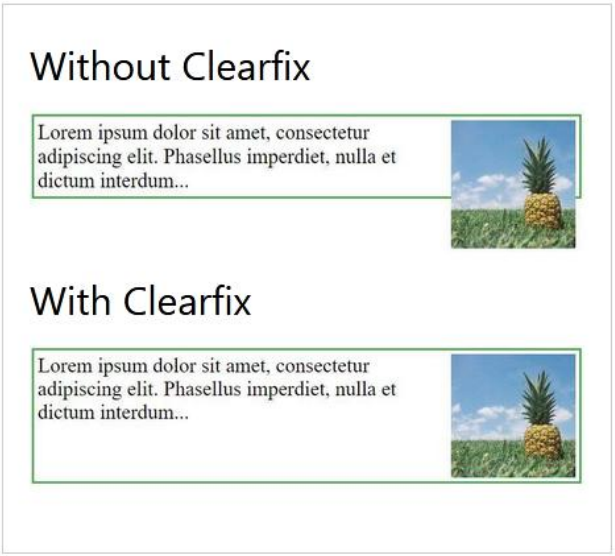
Another method for aligning elements is to use the **float** property:

Example

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

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

The clearfix Hack



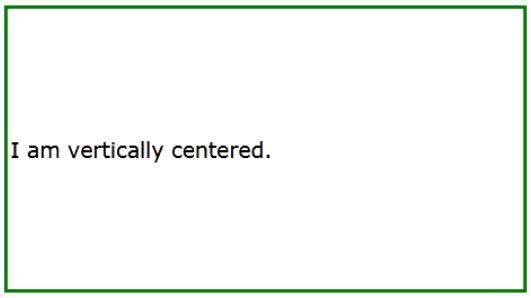
Then we can add **overflow: auto;** to the containing element to fix this problem:

Example

```
.clearfix {
  overflow: auto;
}
```

Center Vertically - Using padding

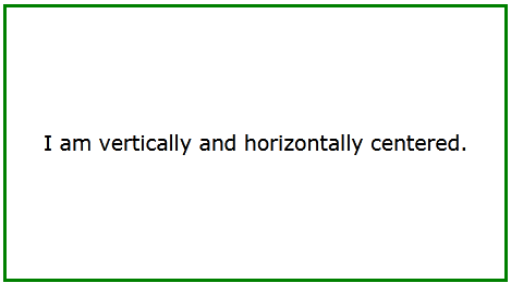
There are many ways to center an element vertically in CSS. A simple solution is to use top and bottom **padding**:



Example

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

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

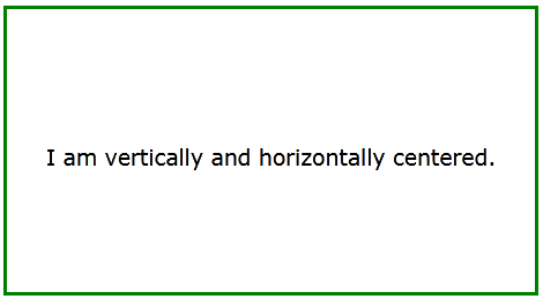


Example

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

Center Vertically - Using line-height

Another trick is to use the **line-height** property with a value that is equal to the **height** property.

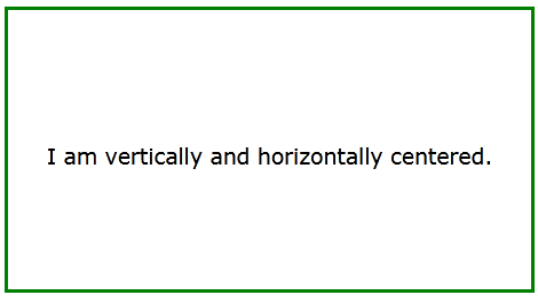


Example

```
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}

/* If the text has multiple lines, add the
following: */
.center p {
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
```

Center Vertically - Using position & transform

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



Example

```
.center {
  height: 200px;
  position: relative;
  border: 3px solid green;
}

.center p {
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

CSS Combinators

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

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

There are four different combinators in CSS:

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

Descendant Selector

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

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

Example

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

Child Selector

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

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

Example

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

Adjacent Sibling Selector

The adjacent sibling selector selects all elements that are the adjacent siblings of a specified element.

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

The following example selects all <p> elements that are placed immediately after elements:

Example

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

General Sibling Selector

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

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

Example

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

All CSS Combinator Selectors

Selector	Example	Example description
element element	div p	Selects all <p> elements inside elements
element>element	div > p	Selects all <p> elements where the parent is a element
element+element	div + p	Selects all <p> elements that are placed immediately after elements

element1~element2	p ~ ul	Selects every element that are preceded by a <p> element

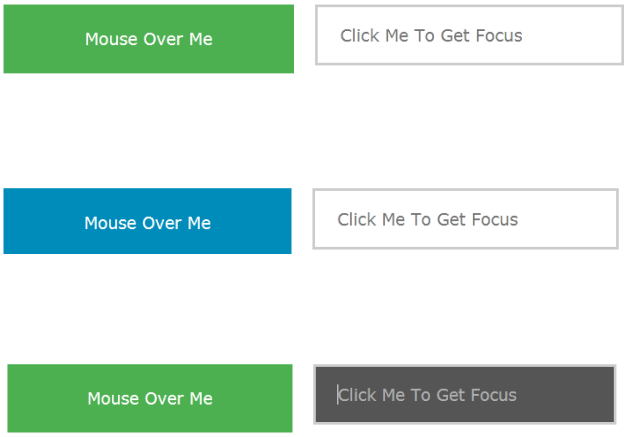
CSS Pseudo-classes

What are Pseudo-classes?

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

For example, it can be used to:

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



Syntax

The syntax of pseudo-classes:

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

Anchor Pseudo-classes

Links can be displayed in different ways:

Example

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}

/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

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:

Example

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

Hover on

An example of using the **:hover** pseudo-class on a element:

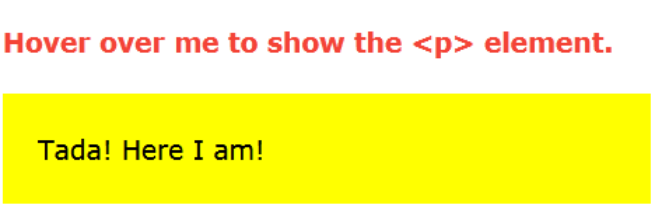
Example

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

Simple Tooltip Hover

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

Hover over me to show the <p> element.



Example

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

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

CSS - The :first-child Pseudo-class

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

Match the first <p> element

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

Example

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

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

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

Example

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

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

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

Example

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

CSS - The :lang Pseudo-class

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

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

Example

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

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

</body>
</html>
```

More Examples

Add different styles to hyperlinks
This example demonstrates how to add other styles to hyperlinks.

Use of :focus
This example demonstrates how to use the :focus pseudo-class.

All CSS Pseudo Classes

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"
:last-child	p:last-child	Selects every <p> elements that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute specified
:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that

		anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

All CSS Pseudo Elements

Selector	Example	Example description
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::selection	p::selection	Selects the portion of an element that is selected by a user

CSS 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:

Example

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

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

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

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

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

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

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

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

The ::first-letter Pseudo-element

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

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

Example

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

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

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

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

Pseudo-elements and CSS Classes

Pseudo-elements can be combined with CSS classes:

Example

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

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

Multiple Pseudo-elements

Several pseudo-elements can also be combined.

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

Example

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

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

CSS - The ::before Pseudo-element

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

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

Example

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

CSS - The ::after Pseudo-element

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

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

Example

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

CSS - The ::selection Pseudo-element

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

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

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

Example

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

All CSS Pseudo Elements

Selector	Example	Example description
::after	p::after	Insert something after the content of each <p> element
::before	p::before	Insert something before the content of each <p> element
::first-letter	p::first-letter	Selects the first letter of each <p> element
::first-line	p::first-line	Selects the first line of each <p> element
::selection	p::selection	Selects the portion of an element that is selected by a user

All CSS Pseudo Classes

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> element of its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a specified range
:invalid	input:invalid	Selects all <input> elements with an invalid value
:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"
:last-child	p:last-child	Selects every <p> elements that is the last child of its parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> element of its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> element
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second child of its parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second child of its parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> element of its parent
:only-child	p:only-child	Selects every <p> element that is the only child of its parent
:optional	input:optional	Selects <input> elements with no "required" attribute
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a specified range
:read-only	input:read-only	Selects <input> elements with a "readonly" attribute

		specified
:read-write	input:read-write	Selects <input> elements with no "readonly" attribute
:required	input:required	Selects <input> elements with a "required" attribute specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicked on a URL containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

CSS Opacity / Transparency

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

Transparent Image

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

opacity 0.2 opacity 0.5 opacity 1 (default)

Note: IE8 and earlier use **filter:alpha(opacity=x)**. The x can take a value from 0 - 100. A lower value makes the element more transparent.

Example

```
img {
  opacity: 0.5;
}
```

Transparent Hover Effect

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

Example

```
img {
  opacity: 0.5;
}

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

Example explained

The first CSS block is similar to the code in Example 1. In addition, we have added what should happen when a user hovers over one of the images. In this case we want the image to NOT be transparent when the user hovers over it. The CSS for this is **opacity:1**;.

When the mouse pointer moves away from the image, the image will be transparent again.

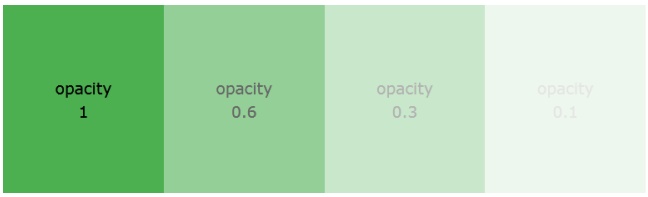
An example of reversed hover effect:

Example

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

Transparent Box

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:

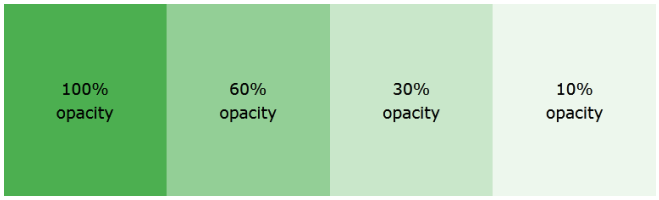


Example

```
div {
  opacity: 0.3;
}
```

Transparency using RGBA

If you do not want to apply opacity to child elements, like in our example above, use **RGBA** color values. The following example sets the opacity for the background color and not the text:



You learned from our CSS Colors Chapter, that you can use RGB as a color value. In addition to RGB, you can use an RGB color value with an alpha channel (RGBA) - which specifies the opacity for a color.

An RGBA color value is specified with: rgba(red, green, blue, *alpha*). The *alpha* parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).

Tip: You will learn more about RGBA Colors in our CSS Colors Chapter.

Example

```
div {
  background: rgba(76, 175, 80, 0.3) /*
Green background with 30% opacity */
}
```

Text in Transparent Box

Example

```
<html>
<head>
<style>
div.background {
  background: url(klematis.jpg) repeat;
  border: 2px solid black;
}

div.transbox {
  margin: 30px;
  background-color: #ffffff;
  border: 1px solid black;
  opacity: 0.6;
}

div.transbox p {
  margin: 5%;
  font-weight: bold;
  color: #000000;
}
</style>
</head>
<body>

<div class="background">
  <div class="transbox">
    <p>This is some text that is placed in the
transparent box.</p>
  </div>
</div>

</body>
</html>
```

Example explained

First, we create a element (class="background") with a background image, and a border.

hen we create another (class="transbox") inside the first .

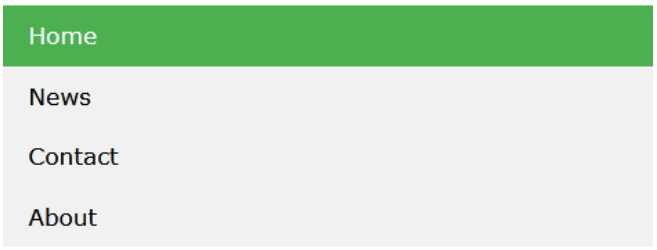
The have a background color, and a border - the div is transparent.

Inside the transparent , we add some text inside a <p> element.

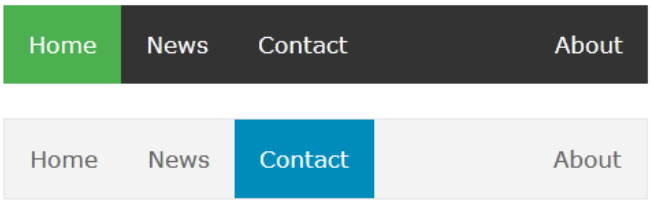
CSS Navigation Bar

Demo: Navigation Bars

Vertical



Horizontal



Navigation Bars

Having easy-to-use navigation is important for any web site.

With CSS you can transform boring HTML menus into good-looking navigation bars.

Navigation Bar = List of Links

A navigation bar needs standard HTML as a base.

In our examples we will build the navigation bar from a standard HTML list.

A navigation bar is basically a list of links, so using the <ul> and <li> elements makes perfect sense:

Example

```
<ul>
  <li><a href="default.asp">Home</a></li>
  <li><a href="news.asp">News</a></li>
  <li><a href="contact.asp">Contact</a></li>
  <li><a href="about.asp">About</a></li>
</ul>
```

Now let's remove the bullets and the margins and padding from the list:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

Example explained:

- **list-style-type: none;** - Removes the bullets. A navigation bar does not need list markers
- Set **margin: 0;** and **padding: 0;** to remove browser default settings

The code in the example above is the standard code used in both vertical, and horizontal navigation bars.

Vertical Navigation Bar

To build a vertical navigation bar, you can style the **<a>** elements inside the list, in addition to the code above:

Example

```
li a {
  display: block;
  width: 60px;
}
```

Example explained:

- **display: block;** - Displaying the links as block elements makes the whole link area clickable (not just the text), and it allows us to specify the width (and padding, margin, height, etc. if you want)
- **width: 60px;** - Block elements take up the full width available by default. We want to specify a 60 pixels width

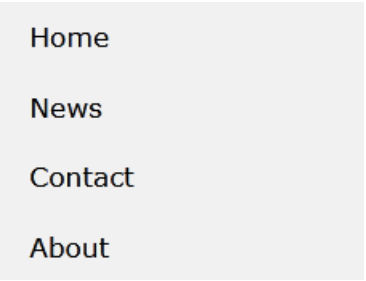
You can also set the width of <ul>, and remove the width of <a>, as they will take up the full width available when displayed as block elements. This will produce the same result as our previous example:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 60px;
}

li a {
  display: block;
}
```

Vertical Navigation Bar Examples

Create a basic vertical navigation bar with a gray background color and change the background color of the links when the user moves the mouse over them:



Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 200px;
  background-color: #f1f1f1;
}

li a {
  display: block;
  color: #000;
  padding: 8px 16px;
  text-decoration: none;
}

/* Change the link color on hover */
li a:hover {
  background-color: #555;
  color: white;
}
```

Active/Current Navigation Link

Add an "active" class to the current link to let the user know which page he/she is on:

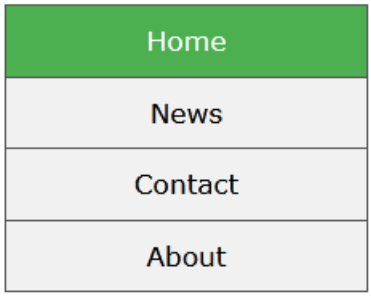
Example

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

Center Links & Add Borders

Add **text-align:center** to <li> or <a> to center the links.

Add the **border** property to <ul> add a border around the navbar. If you also want borders inside the navbar, add a **border-bottom** to all <li> elements, except for the last one:



Example

```
ul {
  border: 1px solid #555;
}

li {
  text-align: center;
  border-bottom: 1px solid #555;
}

li:last-child {
  border-bottom: none;
}
```

Full-height Fixed Vertical Navbar

Create a full-height, "sticky" side navigation:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 25%;
  background-color: #f1f1f1;
  height: 100%; /* Full height */
  position: fixed; /* Make it stick, even on scroll */
  overflow: auto; /* Enable scrolling if the sidenav has too much content */
}
```

Note: This example might not work properly on mobile devices.

Horizontal Navigation Bar

There are two ways to create a horizontal navigation bar. Using **inline** or **floating** list items.

Inline List Items

One way to build a horizontal navigation bar is to specify the <li> elements as inline, in addition to the "standard" code above:

Example

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

Example explained:

- **display: inline;** - By default, <li> elements are block elements. Here, we remove the line breaks before and after each list item, to display them on one line

Floating List Items

Another way of creating a horizontal navigation bar is to float the <li> elements, and specify a layout for the navigation links:

Example

```
li {
  float: left;
}

a {
  display: block;
  padding: 8px;
  background-color: #dddddd;
}
```

Example explained:

- **float: left;** - use float to get block elements to slide next to each other
- **display: block;** - Displaying the links as block elements makes the whole link area clickable (not just the text), and it allows us to specify padding (and height, width, margins, etc. if you want)
- **padding: 8px;** - Since block elements take up the full width available, they cannot float next to each other. Therefore, specify some padding to make them look good
- **background-color: #dddddd;** - Add a gray background-color to each a element

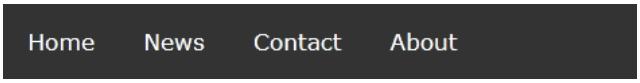
Tip: Add the background-color to <ul> instead of each <a> element if you want a full-width background color:

Example

```
ul {
  background-color: #dddddd;
}
```

Horizontal Navigation Bar Examples

Create a basic horizontal navigation bar with a dark background color and change the background color of the links when the user moves the mouse over them:

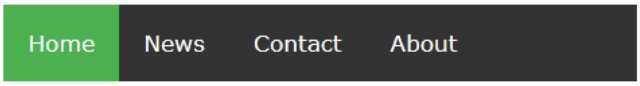


Example

```
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;
}

/* Change the link color to #111 (black) on hover
*/
li a:hover {
  background-color: #111;
}
```

Active/Current Navigation Link

Add an "active" class to the current link to let the user know which page he/she is on:

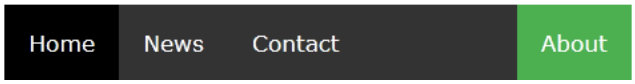


Example

```
.active {
  background-color: #4CAF50;
}
```

Right-Align Links

Right-align links by floating the list items to the right (**float:right;**):

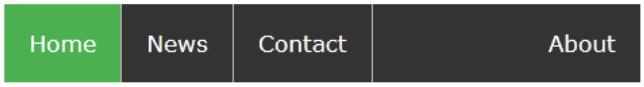


Example

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

Border Dividers

Add the **border-right** property to <li> to create link dividers:



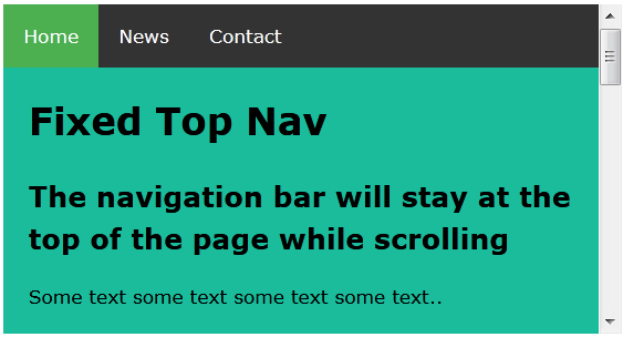
Example

```
/* Add a gray right border to all list items,
except the last item (last-child) */
li {
  border-right: 1px solid #bbb;
}

li:last-child {
  border-right: none;
}
```

e

Fixed Navigation Bar

Make the navigation bar stay at the top or the bottom of the page, even when the user scrolls the page:



Fixed Top

```
ul {
  position: fixed;
  top: 0;
  width: 100%;
}
```

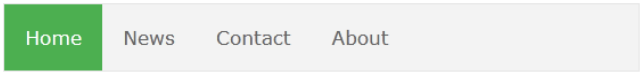
Fixed Bottom

```
ul {
  position: fixed;
  bottom: 0;
  width: 100%;
}
```

Note: Fixed position might not work properly on mobile devices.

Gray Horizontal Navbar

An example of a gray horizontal navigation bar with a thin gray border:



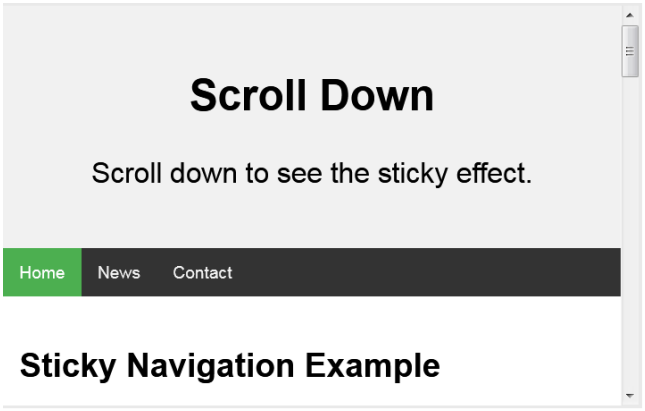
Example

```
ul {
  border: 1px solid #e7e7e7;
  background-color: #f3f3f3;
}

li a {
  color: #666;
}
```

Sticky Navbar

Add **position: sticky;** to <ul> to create a sticky navbar.

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).

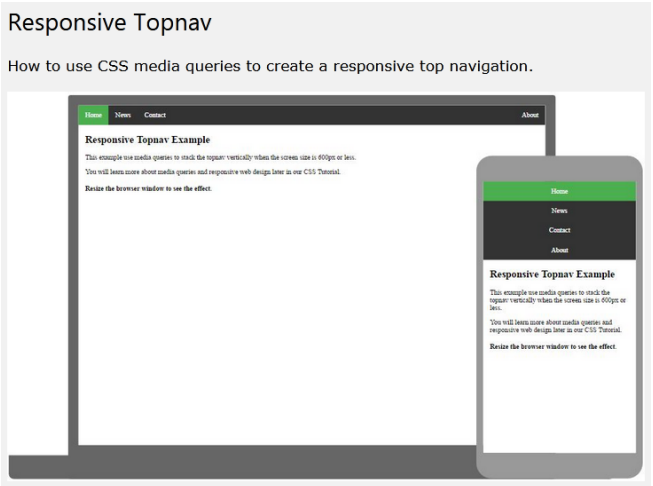


Example

```
ul {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
}
```

Note: Internet Explorer, Edge 15 and earlier versions do not support sticky positioning. Safari requires a -webkit- prefix (see example above). You must also specify at least one of **top**, **right**, **bottom** or **left** for sticky positioning to work.

More Examples

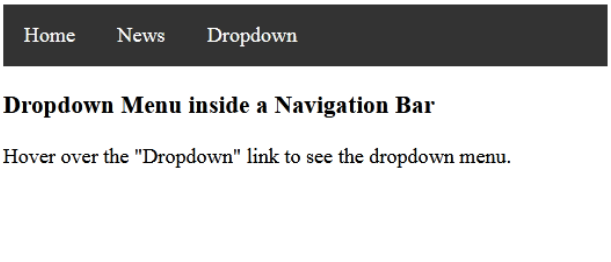


Responsive Sidenav

How to use CSS media queries to create a responsive side navigation.

Dropdown Navbar

How to add a dropdown menu inside a navigation bar.

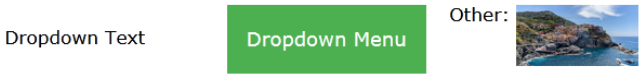


CSS Dropdowns

Create a hoverable dropdown with CSS.

Demo: Dropdown Examples

Move the mouse over the examples below:



Basic Dropdown

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

Example

```
<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>

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

Example Explained

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

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

Wrap a ** ** 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**).

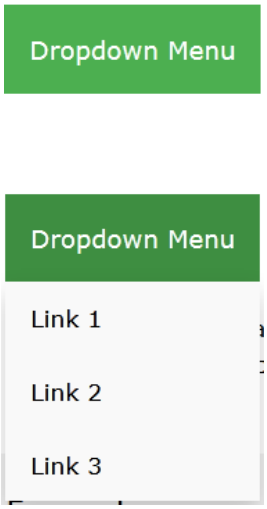
The **.dropdown-content** class holds the actual dropdown content. It is hidden by default, and will be displayed on hover (see below). Note the **min-width** is set to 160px. Feel free to change this. **Tip:** If you want the width of the dropdown content to be as wide as the dropdown button, set the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

Dropdown Menu

Create a dropdown menu that allows the user to choose an option from a list:

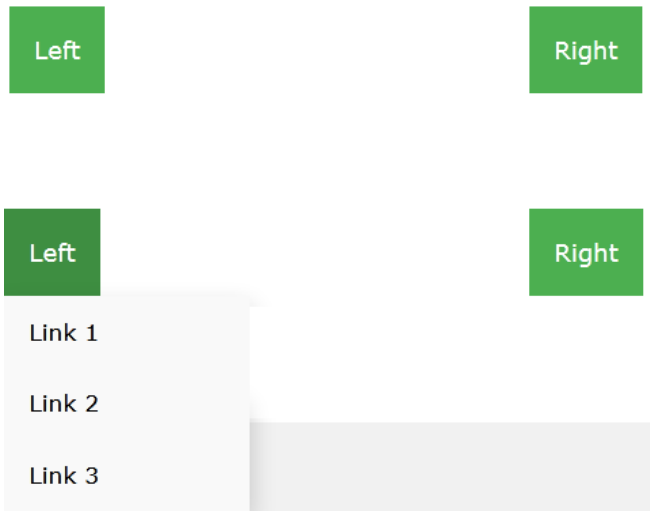


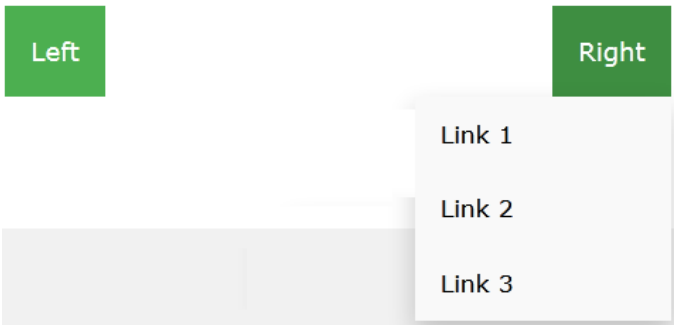
This example is similar to the previous one, except that we add links inside the dropdown box and style them to fit a styled dropdown button:

Example

```
<style>
/* Style The Dropdown Button */
.dropbtn {
  background-color: #4CAF50;
  color: white;
  padding: 16px;
  font-size: 16px;
  border: none;
  cursor: pointer;
}

/* The container <div> - needed to position the dropdown content */
.dropdown {
  position: relative;
  display: inline-block;
}

/* Dropdown Content (Hidden by Default) */
.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;
}

/* Links inside the dropdown */
.dropdown-content a {
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
}

/* Change color of dropdown links on hover */
.dropdown-content a:hover {background-color: #f1f1f1}

/* Show the dropdown menu on hover */
.dropdown:hover .dropdown-content {
  display: block;
}

/* Change the background color of the dropdown button when the dropdown content is shown */
.dropdown:hover .dropbtn {
  background-color: #3e8e41;
}
</style>

<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>
```

Right-aligned Dropdown Content





If you want the dropdown menu to go from right to left, instead of left to right, add **right: 0;**

Example

```
.dropdown-content {
  right: 0;
}
```

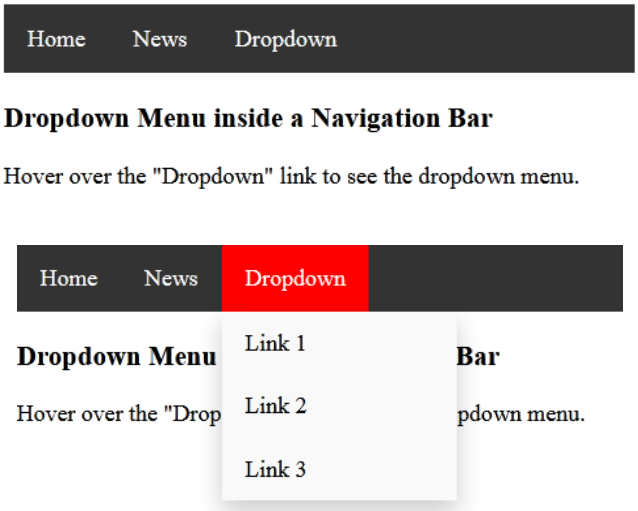
More Examples

Dropdown Image

How to add an image and other content inside the dropdown box

Dropdown Navbar

How to add a dropdown menu inside a navigation bar.



CSS Image Gallery

CSS can be used to create an image gallery.

Image Gallery

The following image gallery is created with CSS:

Example

```
<html>
<head>
<style>
div.gallery {
  margin: 5px;
  border: 1px solid #ccc;
  float: left;
  width: 180px;
}

div.gallery:hover {
  border: 1px solid #777;
}

div.gallery img {
  width: 100%;
  height: auto;
}

div.desc {
  padding: 15px;
  text-align: center;
}
</style>
</head>
```

```
<body>

<div class="gallery">
  <a target="_blank" href="img_5terre.jpg">
    <img src="img_5terre.jpg" alt="Cinque Terre" width="600"
height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_forest.jpg">
    <img src="img_forest.jpg" alt="Forest" width="600" height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_lights.jpg">
    <img src="img_lights.jpg" alt="Northern Lights" width="600"
height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

<div class="gallery">
  <a target="_blank" href="img_mountains.jpg">
    <img src="img_mountains.jpg" alt="Mountains" width="600"
height="400">
  </a>
  <div class="desc">Add a description of the image here</div>
</div>

</body>
</html>
```

More Examples

Responsive Image Gallery

How to use CSS media queries to create a responsive image gallery that will look good on desktops, tablets and smart phones.

CSS Image Sprites

Image Sprites

An image sprite is a collection of images put into a single image.

A web page with many images can take a long time to load and generates multiple server requests.

Using image sprites will reduce the number of server requests and save bandwidth.

Image Sprites - Simple Example

Instead of using three separate images, we use this single image ("img_navsprites.gif"):

With CSS, we can show just the part of the image we need.

In the following example the CSS specifies which part of the "img_navsprites.gif" image to show:

Example

```
#home {
  width: 46px;
  height: 44px;
  background: url(img_navsprites.gif) 0 0;
}
```

Example explained:

- **<img id="home" src="img_trans.gif">** - Only defines a small transparent image because the src attribute cannot be empty. The displayed image will be the background image we specify in CSS
- **width: 46px; height: 44px;** - Defines the portion of the image we want to use
- **background: url(img_navsprites.gif) 0 0;** - Defines the background image and its position (left 0px, top 0px)

This is the easiest way to use image sprites, now we want to expand it by using links and hover effects.

Image Sprites - Create a Navigation List

We want to use the sprite image ("img_navsprites.gif") to create a navigation list.

We will use an HTML list, because it can be a link and also supports a background image:

Example

```
#navlist {
  position: relative;
}

#navlist li {
  margin: 0;
  padding: 0;
  list-style: none;
  position: absolute;
  top: 0;
}

#navlist li, #navlist a {
  height: 44px;
  display: block;
}

#home {
  left: 0px;
  width: 46px;
  background: url('img_navsprites.gif') 0 0;
}

#prev {
  left: 63px;
  width: 43px;
  background: url('img_navsprites.gif') -47px 0;
}

#next {
  left: 129px;
  width: 43px;
  background: url('img_navsprites.gif') -91px 0;
}
```

Example explained:

- #navlist {position:relative;} - position is set to relative to allow absolute positioning inside it
- #navlist li {margin:0;padding:0;list-style:none;position:absolute;top:0;} - margin and padding are set to 0, list-style is removed, and all list items are absolute positioned
- #navlist li, #navlist a {height:44px;display:block;} - the height of all the images are 44px

Now start to position and style for each specific part:

- #home {left:0px;width:46px;} - Positioned all the way to the left, and the width of the image is 46px
- #home {background:url(img_navsprites.gif) 0 0;} - Defines the background image and its position (left 0px, top 0px)
- #prev {left:63px;width:43px;} - Positioned 63px to the right (#home width 46px + some extra space between items), and the width is 43px.
- #prev {background:url('img_navsprites.gif') -47px 0;} - Defines the background image 47px to the right (#home width 46px + 1px line divider)
- #next {left:129px;width:43px;}- Positioned 129px to the right (start of #prev is 63px + #prev width 43px + extra space), and the width is 43px.
- #next {background:url('img_navsprites.gif') -91px 0;} - Defines the background image 91px to the right (#home width 46px + 1px line divider + #prev width 43px + 1px line divider)

Image Sprites - Hover Effect

Now we want to add a hover effect to our navigation list.

Tip: The **:hover** selector can be used on all elements, not only on links.

Our new image ("img_navsprites_hover.gif") contains three navigation images and three images to use for hover effects:

Because this is one single image, and not six separate files, there will be **no loading delay** when a user hovers over the image.

We only add three lines of code to add the hover effect:

Example

```
#home a:hover {
  background: url('img_navsprites_hover.gif') 0
-45px;
}

#prev a:hover {
  background: url('img_navsprites_hover.gif')
-47px -45px;
}

#next a:hover {
  background: url('img_navsprites_hover.gif')
-91px -45px;
}
```

Example explained:

- #home a:hover {background: transparent url('img_navsprites_hover.gif') 0 -45px;} - For all three hover images we specify the same background position, only 45px further down

CSS 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:

Example

```
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:

Example

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

CSS [attribute~="value"] Selector

The **[attribute~="value"]** selector is used to select elements with an attribute value containing a specified word.

The following example selects all elements with a title attribute that contains a space-separated list of words, one of which is "flower":

Example

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

The example above will match elements with title="flower", title="summer flower", and title="flower new", but not title="my-flower" or title="flowers".

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"!

Example

```
[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!

Example

```
[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!

Example

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

CSS [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!

Example

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

Styling Forms

The attribute selectors can be useful for styling forms without class or ID:

Example

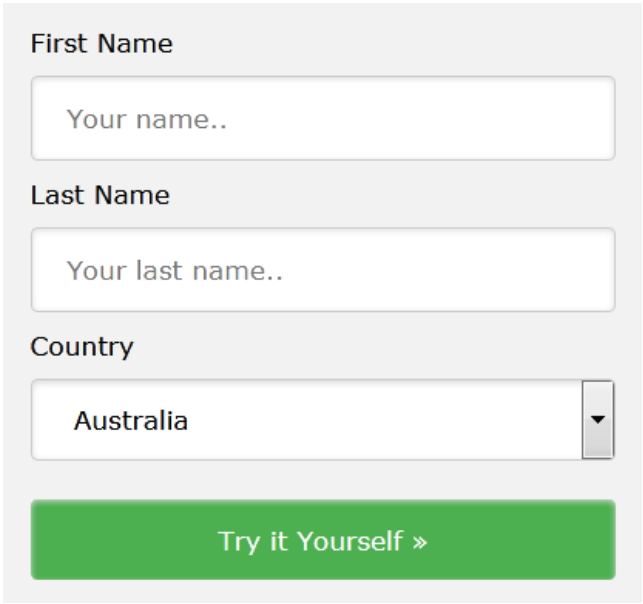
```
input[type="text"] {
  width: 150px;
  display: block;
  margin-bottom: 10px;
  background-color: yellow;
}

input[type="button"] {
  width: 120px;
  margin-left: 35px;
  display: block;
}
```

All CSS Attribute Selectors

Selector	Example	Example description
[attribute]	[target]	Selects all elements with a target attribute
[attribute=value]	[target=_blank]	Selects all elements with target="_blank"
[attribute~=value]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"
[attribute|=value]	[lang|=en]	Selects all elements with a lang attribute value starting with "en"
[attribute^=value]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"
[attribute$=value]	a[href$=".pdf"]	Selects every <a> element whose href attribute value ends with ".pdf"
[attribute=value]*	a[href*="w3schools"]	Selects every <a> element whose href attribute value contains the substring "w3schools"

CSS Forms

The look of an HTML form can be greatly improved with CSS:



Styling Input Fields

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

Example

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

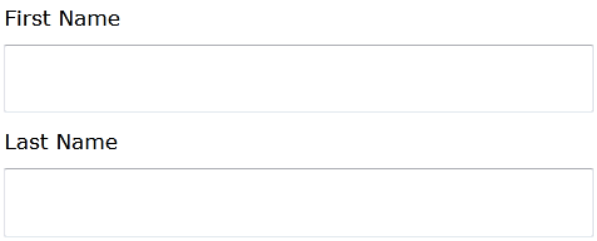
The example above applies to all <input> elements. If you only want to style a specific input type, you can use attribute selectors:

- **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:



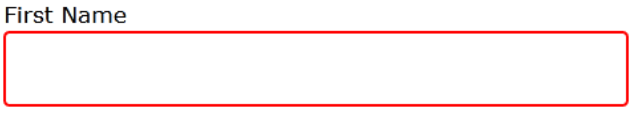
Example

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

Note that we have set the **box-sizing** property to **border-box**. This makes sure that the padding and eventually borders are included in the total width and height of the elements. Read more about the **box-sizing** property in our CSS Box Sizing chapter.

Bordered Inputs

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



Example

```
input[type=text] {
  border: 2px solid red;
  border-radius: 4px;
}
```

If you only want a bottom border, use the **border-bottom** property:

Example

```
input[type=text] {
  border: none;
  border-bottom: 2px solid red;
}
```

Colored Inputs

Use the **background-color** property to add a background color to the input, and the **color** property to change the text color:

Example

```
input[type=text] {
  background-color: #3CBC8D;
  color: white;
}
```

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:

Example

```
input[type=text]:focus {
  background-color: lightblue;
}
```


Example

```
input[type=text]:focus {
  border: 3px solid #555;
}
```

Input with icon/image

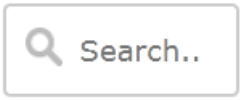
If you want an icon inside the input, use the **background-image** property and position it with the **background-position** property. Also notice that we add a large left padding to reserve the space of the icon:

Example

```
input[type=text] {
  background-color: white;
  background-image: url('searchicon.png');
  background-position: 10px 10px;
  background-repeat: no-repeat;
  padding-left: 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.



Example

```
input[type=text] {
  transition: width 0.4s ease-in-out;
}

input[type=text]:focus {
  width: 100%;
}
```

Styling Textareas

Tip: Use the **resize** property to prevent textareas from being resized (disable the "grabber" in the bottom right corner):

Some text...

Example

```
textarea {
  width: 100%;
  height: 150px;
  padding: 12px 20px;
  box-sizing: border-box;
  border: 2px solid #ccc;
  border-radius: 4px;
  background-color: #f8f8f8;
  resize: none;
}
```

Styling Select Menus

Australia

Example

```
select {
  width: 100%;
  padding: 16px 20px;
  border: none;
  border-radius: 4px;
  background-color: #f1f1f1;
}
```

Styling Input Buttons

Button

Button

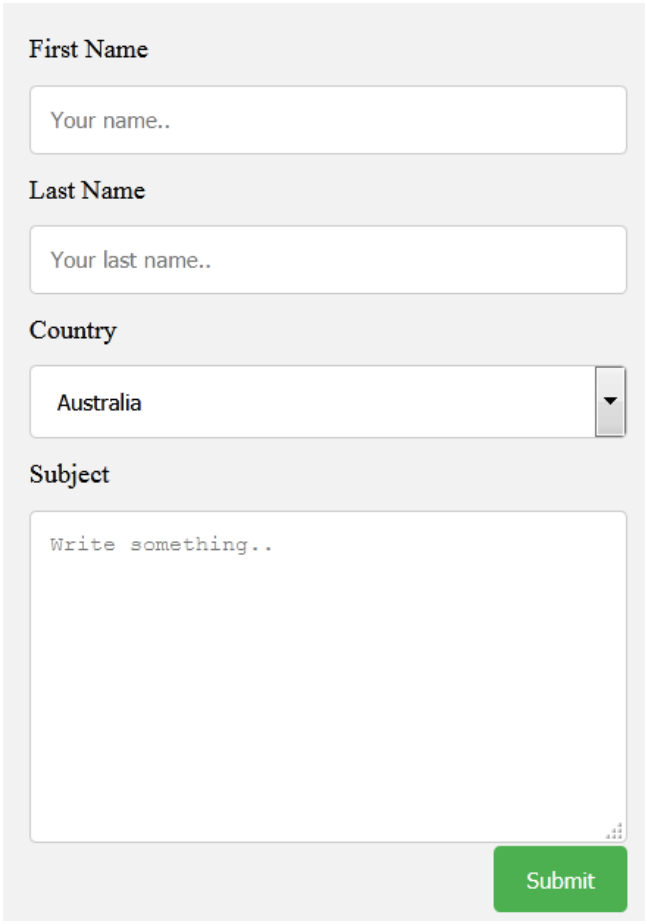
Example

```
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;
}

/* Tip: use width: 100% for full-width buttons */
```

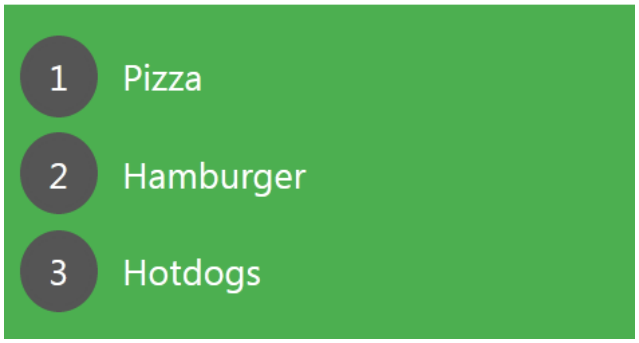
Responsive Form

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.



CSS Counters



CSS counters are "variables" maintained by CSS whose values can be incremented by CSS rules (to track how many times they are used). Counters let you adjust the appearance of content based on its placement in the document.

Automatic Numbering With Counters

CSS counters are like "variables". The variable values can be incremented by CSS rules (which will track how many times they are used).

To work with CSS counters we will use the following properties:

- **counter-reset** - Creates or resets a counter
- **counter-increment** - Increments a counter value
- **content** - Inserts generated content
- **counter()** or **counters()** function - Adds the value of a counter to an element

To use a CSS counter, it must first be created with **counter-reset**.

The following example creates a counter for the page (in the body selector), then increments the counter value for each **<h2>** element and adds "**Section *<value of the counter>*:**" to the beginning of each **<h2>** element:

Example

```
body {
  counter-reset: section;
}

h2::before {
  counter-increment: section;
  content: "Section " counter(section) ": ";
}
```

Nesting Counters

The following example creates one counter for the page (section) and one counter for each **<h1>** element (subsection). The "section" counter will be counted for each **<h1>** element with "Section ***<value of the section counter>***.", and the "subsection" counter will be counted for each **<h2>** element with "***<value of the section counter>***.***<value of the subsection counter>***":

Example

```
body {
  counter-reset: section;
}

h1 {
  counter-reset: subsection;
}

h1::before {
  counter-increment: section;
  content: "Section " counter(section) ". ";
}

h2::before {
  counter-increment: subsection;
  content: counter(section) "."
counter(subsection) " ";
}
```

A counter can also be useful to make outlined lists because a new instance of a counter is automatically created in child elements. Here we use the **counters()** function to insert a string between different levels of nested counters:

Example

```
ol {
  counter-reset: section;
  list-style-type: none;
}

li::before {
  counter-increment: section;
  content: counters(section,".") " ";
}
```

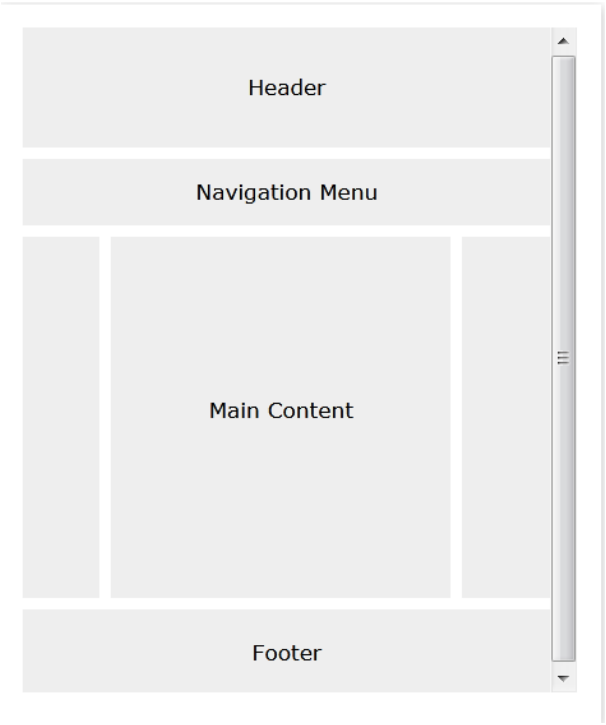
CSS Counter Properties

Property	Description
content	Used with the ::before and ::after pseudo-elements, to insert generated content
counter-increment	Increments one or more counter values
counter-reset	Creates or resets one or more counters

CSS Website Layout

Website Layout

A website is often divided into headers, menus, content and a footer:



There are tons of different layout designs to choose from. However, the structure above, is one of the most common, and we will take a closer look at it in this tutorial.

Header

A header is usually located at the top of the website (or right below a top navigation menu). It often contains a logo or the website name:

Example

```
.header {
  background-color: #F1F1F1;
  text-align: center;
  padding: 20px;
}
```

Result

Header

Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website:

Example

```
/* The navbar container */
.topnav {
  overflow: hidden;
  background-color: #333;
}

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

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

Result

Link Link Link

Content

The layout in this section, often depends on the target users. The most common layout is one (or combining them) of the following:

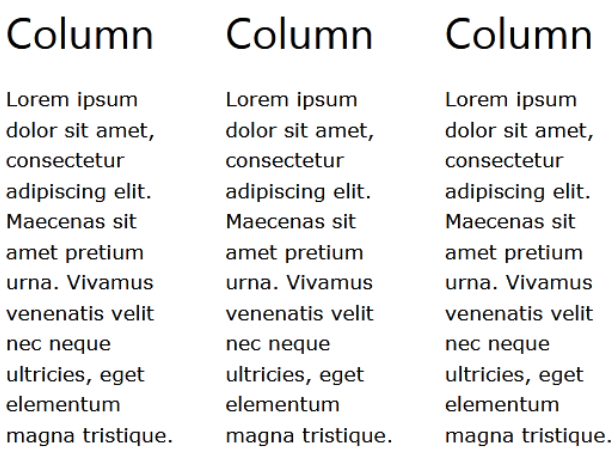
- **1-column** (often used for mobile browsers)
- **2-column** (often used for tablets and laptops)
- **3-column layout** (only used for desktops)

We will create a 3-column layout, and change it to a 1-column layout on smaller screens:

Example

```
/* Create three equal columns that floats next to
each other */
.column {
  float: left;
  width: 33.33%;
}

/* Clear floats after the columns */
.row:after {
  content: "";
  display: table;
  clear: both;
}

/* Responsive layout - makes the three columns
stack on top of each other instead of next to
each other on smaller screens (600px wide or
less) */
@media screen and (max-width: 600px) {
  .column {
    width: 100%;
  }
}
```

Result



Tip: To create a 2-column layout, change the width to 50%. To create a 4-column layout, use 25%, etc.

Tip: Do you wonder how the @media rule works? Read more about it in our CSS Media Queries chapter.

Tip: A more modern way of creating column layouts, is to use CSS Flexbox. However, it is not supported in Internet Explorer 10 and earlier versions. If you require IE6-10 support, use floats (as shown above).

To learn more about the Flexible Box Layout Module, read our CSS Flexbox chapter.

Unequal Columns

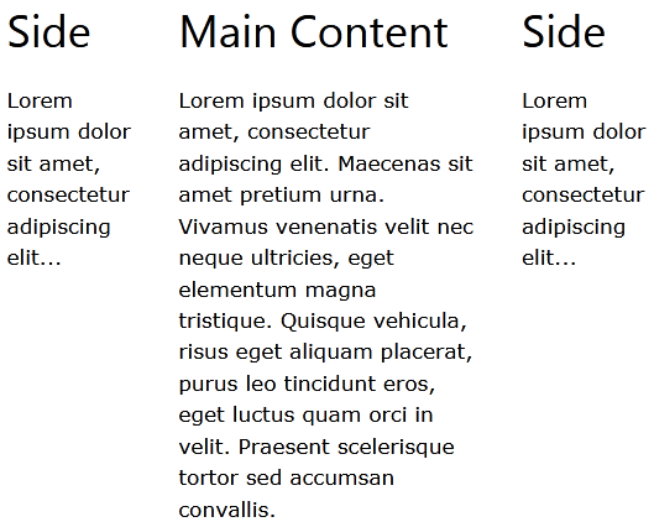
The main content is the biggest and the most important part of your site.

It is common with **unequal** column widths, so that most of the space is reserved for the main content. The side content (if any) is often used as an alternative navigation or to specify information relevant to the main content. Change the widths as you like, only remember that it should add up to 100% in total:

Example

```
.column {
  float: left;
}

/* Left and right column */
.column.side {
  width: 25%;
}

/* Middle column */
.column.middle {
  width: 50%;
}

/* Responsive layout - makes the three columns stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .column.side, .column.middle {
    width: 100%;
  }
}
```

Result



Footer

The footer is placed at the bottom of your page. It often contains information like copyright and contact info:

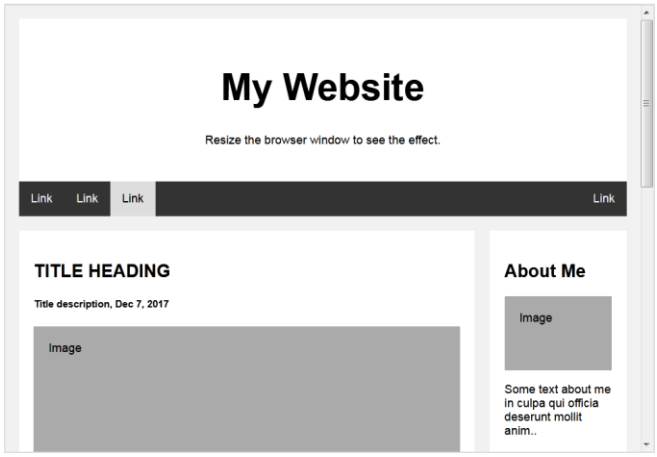
Example

```
.footer {
  background-color: #F1F1F1;
  text-align: center;
  padding: 10px;
}
```

Result

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:



CSS Units

CSS Units

CSS has several different units for expressing a length.

Many CSS properties take "length" values, such as **width**, **margin**, **padding**, **font-size**, etc.

Length is a number followed by a length unit, such as 10px, 2em, etc.

A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted.

For some CSS properties, negative lengths are allowed.

There are two types of length units: absolute and relative.

Absolute Lengths

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

Unit	Description	
cm	centimeters	Try it
mm	millimeters	Try it
in	inches (1in = 96px = 2.54cm)	Try it
px *	pixels (1px = 1/96th of 1in)	Try it
pt	points (1pt = 1/72 of 1in)	Try it
pc	picas (1pc = 12 pt)	Try it

* Pixels (px) are relative to the viewing device. For low-dpi devices, 1px is one device pixel (dot) of the display. For printers and high resolution screens 1px implies multiple device pixels.

Relative Lengths

Relative length units specify a length relative to another length property. Relative length units scales better between different rendering mediums.

Unit	Description	
em	Relative to the font-size of the element (2em means 2 times the size of the current font)	Try it
ex	Relative to the x-height of the current font (rarely used)	Try it
ch	Relative to width of the "0" (zero)	Try it
rem	Relative to font-size of the root element	Try it
vw	Relative to 1% of the width of the viewport*	Try it
vh	Relative to 1% of the height of the viewport*	Try it
vmin	Relative to 1% of viewport's* smaller dimension	Try it
vmax	Relative to 1% of viewport's* larger dimension	Try it
%	Relative to the parent element	Try it

Tip: The em and rem units are practical in creating perfectly scalable layout!
* Viewport = the browser window size. If the viewport is 50cm wide, 1vw = 0.5cm.

Browser Support

The numbers in the table specify the first browser version that fully supports the length unit.

Length Unit					
em, ex, %, px, cm, mm, in, pt, pc	1.0	3.0	1.0	1.0	3.5
ch	27.0	9.0	1.0	7.0	20.0
rem	4.0	9.0	3.6	4.1	11.6
vh, vw	20.0	9.0	19.0	6.0	20.0
vmin	20.0	9.0*	19.0	6.0	20.0
vmax	26.0	Not supported	19.0	7.0	20.0

Note: Internet Explorer 9 supports vmin with the non-standard name: vm.

CSS Specificity

What is Specificity?

If there are two or more conflicting CSS rules that point to the same element, the browser follows some rules to determine which one is most specific and therefore wins out.

Think of specificity as a score/rank that determines which style declarations are ultimately applied to an element.

The universal selector (*) has low specificity, while ID selectors are highly specific!

Note: Specificity is a common reason why your CSS-rules don't apply to some elements, although you think they should.

Specificity Hierarchy

Every selector has its place in the specificity hierarchy. There are four categories which define the specificity level of a selector:

Inline styles - An inline style is attached directly to the element to be styled. Example: <h1 style="color: #ffffff;">.

IDs - An ID is a unique identifier for the page elements, such as #navbar.

Classes, attributes and pseudo-classes - This category includes .classes, [attributes] and pseudo-classes such as :hover, :focus etc.

Elements and pseudo-elements - This category includes element names and pseudo-elements, such as h1, div, :before and :after.

How to Calculate Specificity?

Memorize how to calculate specificity!

Start at 0, add 1000 for style attribute, add 100 for each ID, add 10 for each attribute, class or pseudo-class, add 1 for each element name or pseudo-element.

Consider these three code fragments:

Example

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

The specificity of A is 1 (one element)
The specificity of B is 101 (one ID reference and one element)
The specificity of C is 1000 (inline styling)

Since 1 < 101 < 1000, the third rule (C) has a greater level of specificity, and therefore will be applied.

Specificity Rules

Equal specificity: the latest rule counts - If the same rule is written twice into the external style sheet, then the lower rule in the style sheet is closer to the element to be styled, and therefore will be applied:

Example

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

the latter rule is always applied.

ID selectors have a higher specificity than attribute selectors - Look at the following three code lines:

Example

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

the first rule is more specific than the other two, and will be applied.

Contextual selectors are more specific than a single element selector - The embedded style sheet is closer to the element to be styled. So in the following situation

Example

```
From external CSS file:
#content h1 {background-color: red;}

In HTML file:
<style>
#content h1 {
  background-color: yellow;
}
</style>
```

the latter rule will be applied.

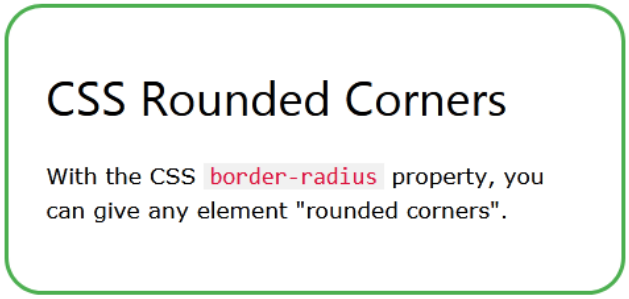
A class selector beats any number of element selectors - a class selector such as .intro beats h1, p, div, etc:

Example

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

The universal selector and inherited values have a specificity of 0 - *, body * and similar have a zero specificity. Inherited values also have a specificity of 0.

CSS Rounded Corners



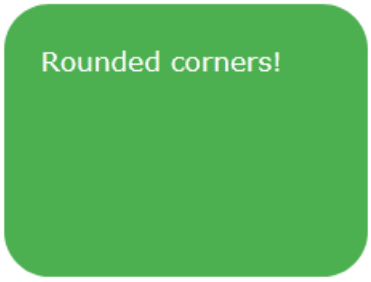
CSS border-radius Property

The CSS border-radius property defines the radius of an element's corners.

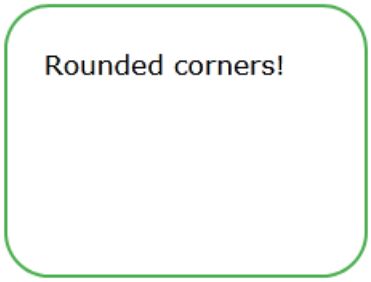
Tip: This property allows you to add rounded corners to elements!

Here are three examples:

1. Rounded corners for an element with a specified background color:



2. Rounded corners for an element with a border:



3. Rounded corners for an element with a background image:

Here is the code:

Example

```
#rcorners1 {
  border-radius: 25px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners2 {
  border-radius: 25px;
  border: 2px solid #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners3 {
  border-radius: 25px;
  background: url(paper.gif);
  background-position: left top;
  background-repeat: repeat;
  padding: 20px;
  width: 200px;
  height: 150px;
}
```

Tip: The **border-radius** property is actually a shorthand property for the **border-top-left-radius**, **border-top-right-radius**, **border-bottom-right-radius** and **border-bottom-left-radius** properties.

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:

Example

```
#rcorners1 {
  border-radius: 15px 50px 30px 5px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners2 {
  border-radius: 15px 50px 30px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners3 {
  border-radius: 15px 50px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners4 {
  border-radius: 15px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}
```

You could also create elliptical corners:

Example

```
#rcorners1 {
  border-radius: 50px / 15px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners2 {
  border-radius: 15px / 50px;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}

#rcorners3 {
  border-radius: 50%;
  background: #73AD21;
  padding: 20px;
  width: 200px;
  height: 150px;
}
```

CSS Rounded Corners Properties

Property	Description
border-radius	A shorthand property for setting all the four border-*-*-radius properties
border-top-left-radius	Defines the shape of the border of the top-left corner
border-top-right-radius	Defines the shape of the border of the top-right corner
border-bottom-right-radius	Defines the shape of the border of the bottom-right corner
border-bottom-left-radius	Defines the shape of the border of the bottom-left corner

CSS Border Images

CSS border-image Property

The CSS **border-image** property allows you to specify an image to be used instead of the normal border around an element.

The property has three parts:

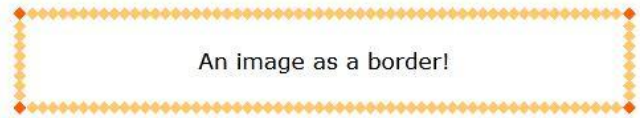
1. The image to use as the border
2. Where to slice the image
3. Define whether the middle sections should be repeated or stretched

We will use the following image (called "border.png"):

The **border-image** property takes the image and slices it into nine sections, like a tic-tac-toe board. It then places the corners at the corners, and the middle sections are repeated or stretched as you specify.

Note: For **border-image** to work, the element also needs the **border** property set!

Here, the middle sections of the image are repeated to create the border:

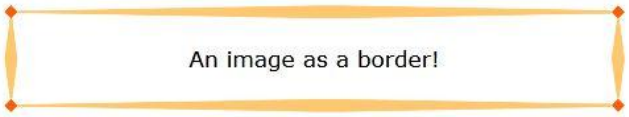


Here is the code:

Example

```
#borderimg {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30 round;
}
```

Here, the middle sections of the image are stretched to create the border:



Here is the code:

Example

```
#borderimg {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30 stretch;
}
```

Tip: The **border-image** property is actually a shorthand property for the **border-image-source**, **border-image-slice**, **border-image-width**, **border-image-outset** and **border-image-repeat** properties.

CSS border-image - Different Slice Values

Different slice values completely changes the look of the border:

Example 1:

Example 2:

Example 3:

Here is the code:

Example

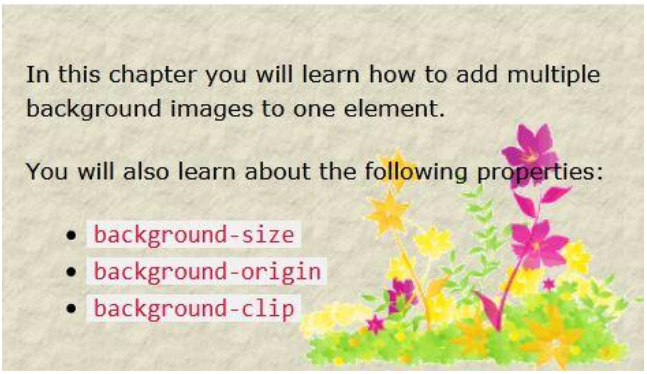
```
#borderimg1 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 50 round;
}

#borderimg2 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 20% round;
}

#borderimg3 {
  border: 10px solid transparent;
  padding: 15px;
  border-image: url(border.png) 30% round;
}
```

CSS Border Image Properties

Property	Description
border-image	A shorthand property for setting all the border-image-* properties
border-image-source	Specifies the path to the image to be used as a border
border-image-slice	Specifies how to slice the border image

border-image-width	Specifies the widths of the border image
border-image-outset	Specifies the amount by which the border image area extends beyond the border box
border-image-repeat	Specifies whether the border image should be repeated, rounded or stretched

CSS Multiple Backgrounds



CSS 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;
}
```

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;
}
```

CSS 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.

The following example resizes a background image to much smaller than the original image (using pixels):

Here is the code:

Example

```
#div1 {
  background: url(img_flower.jpg);
  background-size: 100px 80px;
  background-repeat: no-repeat;
}
```

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**:

Example

```
#div1 {
  background: url(img_flower.jpg);
  background-size: contain;
  background-repeat: no-repeat;
}
#div2 {
  background: url(img_flower.jpg);
  background-size: cover;
  background-repeat: no-repeat;
}
```

Define Sizes of Multiple Background Images

The **background-size** property also accepts multiple values for background size (using a comma-separated list), when working with multiple backgrounds.

The following example has three background images specified, with different background-size value for each image:

Example

```
#example1 {
  background: url(img_tree.gif) left top no-
repeat, url(img_flwr.gif) right bottom no-repeat,
url(paper.gif) left top repeat;
  background-size: 50px, 130px, auto;
}
```

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:

Example

```
.hero-image {
  background: url(img_man.jpg) no-repeat center;
  background-size: cover;
  height: 500px;
  position: relative;
}
```

Hero Image

You could also use different background properties on a to create a hero image (a large image with text), and place it where you want.

Example

```
.hero-image {
  background: url(img_man.jpg) no-repeat center;
  background-size: cover;
  height: 500px;
  position: relative;
}
```

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:

Example

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

Example

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

CSS Advanced Background Properties

Property	Description
background	A shorthand property for setting all the background properties in one declaration
background-clip	Specifies the painting area of the background
background-image	Specifies one or more background images for an element
background-origin	Specifies where the background image(s) is/are positioned
background-size	Specifies the size of the background image(s)

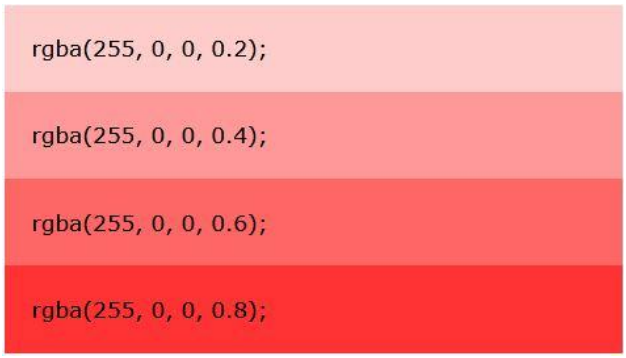
CSS Colors

CSS supports 140+ color names, HEX values, RGB values, RGBA values, HSL values, HSLA values, and opacity.

RGBA Colors

RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color.

An RGBA color value is specified with: rgba(red, green, blue, alpha). The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).



The following example defines different RGBA colors:

Example

```
#p1 {background-color: rgba(255, 0, 0, 0.3);}  /*
red with opacity */
#p2 {background-color: rgba(0, 255, 0, 0.3);}  /*
green with opacity */
#p3 {background-color: rgba(0, 0, 255, 0.3);}  /*
blue with opacity */
```

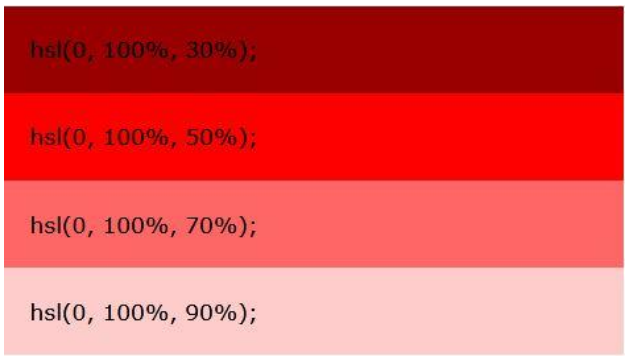
HSL Colors

HSL stands for Hue, Saturation and Lightness.

An HSL color value is specified with: hsl(hue, saturation, lightness).

1. Hue is a degree on the color wheel (from 0 to 360):
   - 0 (or 360) is red
   - 120 is green
   - 240 is blue
2. Saturation is a percentage value: 100% is the full color.

3. Lightness is also a percentage; 0% is dark (black) and 100% is white.



The following example defines different HSL colors:

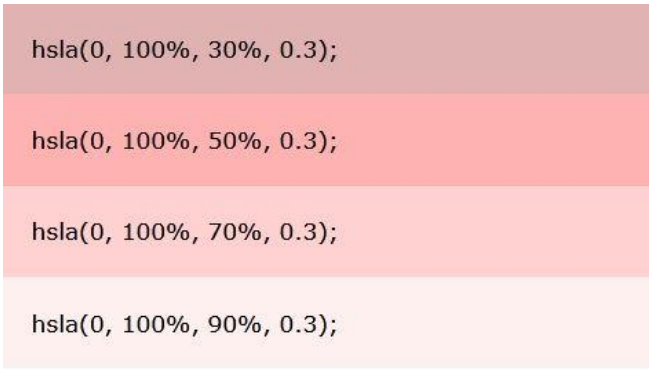
Example

```
#p1 {background-color: hsl(120, 100%, 50%);}  /*
green */
#p2 {background-color: hsl(120, 100%, 75%);}  /*
light green */
#p3 {background-color: hsl(120, 100%, 25%);}  /*
dark green */
#p4 {background-color: hsl(120, 60%, 70%);}   /*
pastel green */
```

HSLA Colors

HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.

An HSLA color value is specified with: hsla(hue, saturation, lightness, alpha), where the alpha parameter defines the opacity. The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (fully opaque).



The following example defines different HSLA colors:

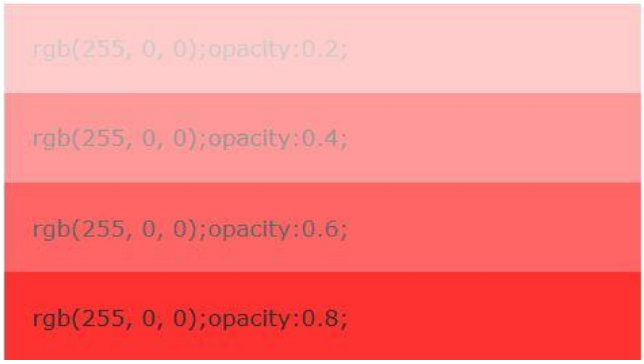
Example

```
#p1 {background-color: hsla(120, 100%, 50%,
0.3);}  /* green with opacity */
#p2 {background-color: hsla(120, 100%, 75%,
0.3);}  /* light green with opacity */
#p3 {background-color: hsla(120, 100%, 25%,
0.3);}  /* dark green with opacity */
#p4 {background-color: hsla(120, 60%, 70%,
0.3);}   /* pastel green with opacity */
```

Opacity

The CSS **opacity** property sets the opacity for the whole element (both background color and text will be opaque/transparent).

The **opacity** property value must be a number between 0.0 (fully transparent) and 1.0 (fully opaque).



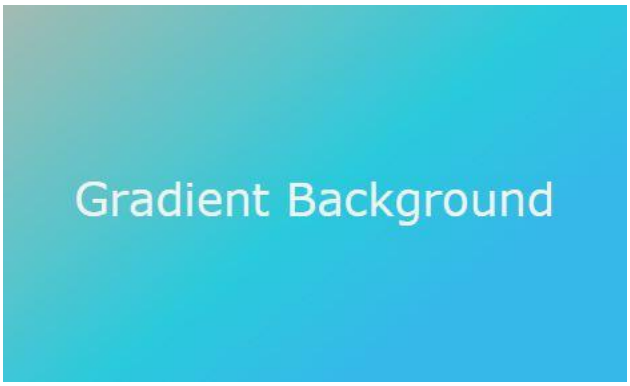
Notice that the text above will also be transparent/opaque!

The following example shows different elements with opacity:

Example

```
#p1 {background-color:rgb(255,0,0);opacity:0.6;}
/* red with opacity */
#p2 {background-color:rgb(0,255,0);opacity:0.6;}
/* green with opacity */
#p3 {background-color:rgb(0,0,255);opacity:0.6;}
/* blue with opacity */
```

CSS 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 - Top to Bottom (this is default)

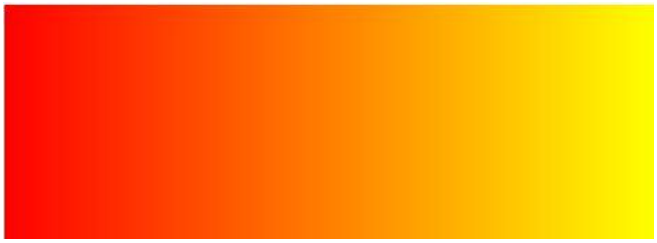
The following example shows a linear gradient that starts at the top. It starts red, transitioning to yellow:

Example

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

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);
}
```

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);
}
```

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);
}
```

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);
}
```

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);
}
```

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).

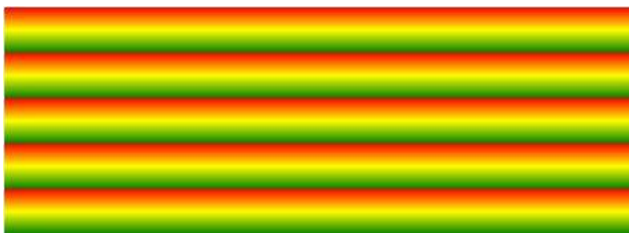
The following example shows a linear gradient that starts from the left. It starts fully transparent, transitioning to full color red:

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%);
}
```

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.

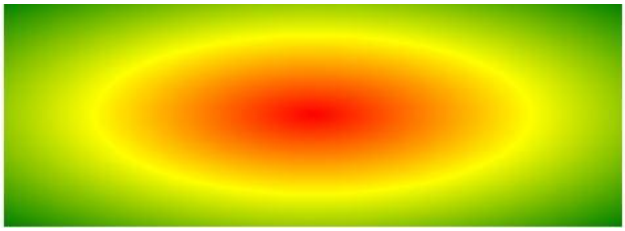
Syntax

```
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:

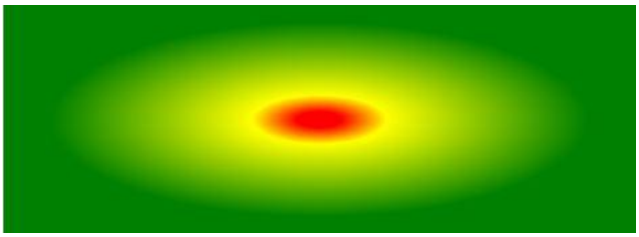


Example

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

Radial Gradient - Differently Spaced Color Stops

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



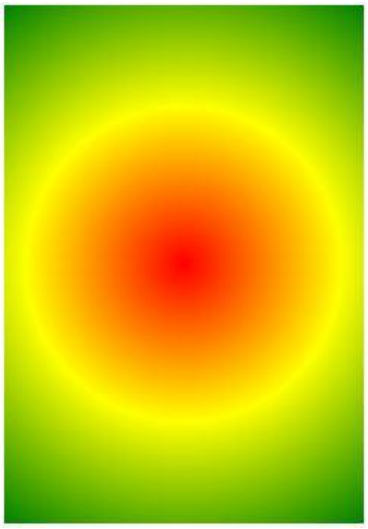
Example

```
#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);
}
```

Use of Different Size Keywords

The size parameter defines the size of the gradient. It can take four values:

- **closest-side**
- **farthest-side**
- **closest-corner**
- **farthest-corner**

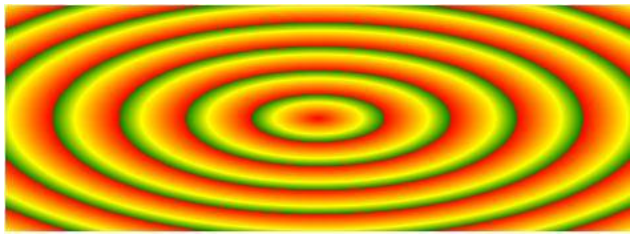
Example

A radial gradient with different size keywords

```
#grad1 {
  background-image: radial-gradient(closest-side
at 60% 55%, red, yellow, black);
}

#grad2 {
  background-image: radial-gradient(farthest-side
at 60% 55%, red, yellow, black);
}
```

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%);
}
```

CSS Gradient Properties

The following table lists the CSS gradient properties:

Property	Description
background-image	Sets one or more background images for an element

CSS Shadow Effects

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**

CSS Text Shadow

The CSS text-shadow property applies 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;
}
```

Next, add a color to the shadow:

Example

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

Then, add a blur effect to the shadow:

Example

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

The following example shows a white text with black shadow:

Example

```
h1 {
  color: white;
  text-shadow: 2px 2px 4px #000000;
}
```

The following example shows a red neon glow shadow:

Example

```
h1 {
  text-shadow: 0 0 3px #FF0000;
}
```

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:

Example

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

The following example shows a white text with black, blue, and darkblue shadow:

Example

```
h1 {
  color: white;
  text-shadow: 1px 1px 2px black, 0 0 25px blue,
0 0 5px darkblue;
}
```

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

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 element with a black box-shadow

Example

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

Next, add a color to the shadow:

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

Example

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

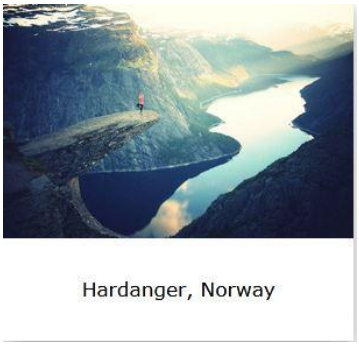
You can also add shadows to the ::before and ::after pseudo-elements, to create an interesting effect:

Example

```
#boxshadow {
  position: relative;
  box-shadow: 1px 2px 4px rgba(0, 0, 0, .5);
  padding: 10px;
  background: white;
}

#boxshadow img {
  width: 100%;
  border: 1px solid #8a4419;
  border-style: inset;
}

#boxshadow::after {
  content: '';
  position: absolute;
  z-index: -1; /* hide shadow behind image */
  box-shadow: 0 15px 20px rgba(0, 0, 0, 0.3);
  width: 70%;
  left: 15%; /* one half of the remaining 30% */
  height: 100px;
  bottom: 0;
}
```

Cards

An example of using the **box-shadow** property to create paper-like cards:



Example

```
div.card {
  width: 250px;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px
20px 0 rgba(0, 0, 0, 0.19);
  text-align: center;
}
```

CSS Shadow Properties

The following table lists the CSS shadow properties:

Property	Description
box-shadow	Adds one or more shadows to an element
text-shadow	Adds one or more shadows to a text

CSS Text Effects

CSS Text Overflow, Word Wrap, Line Breaking Rules, and Writing Modes

In this chapter you will learn about the following properties:

- **text-overflow**
- **word-wrap**
- **word-break**
- **writing-mode**

CSS Text Overflow

The CSS **text-overflow** property specifies how overflowed content that is not displayed should be signaled to the user.

It can be clipped:

This is some long text that

or it can be rendered as an ellipsis (...):

This is some long text t...

The CSS code is as follows:

Example

```
p.test1 {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: clip;
}

p.test2 {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

The following example shows how you can display the overflowed content when hovering over the element:

Example

```
div.test:hover {
  overflow: visible;
}
```

CSS Word Wrapping

The CSS word-wrap property allows long words to be able to be broken and wrap onto the next line.

If a word is too long to fit within an area, it expands outside:

This paragraph contains a very long word: thisisaveryveryveryveryveryverylongword. The long word will break and wrap to the next line.

The word-wrap property allows you to force the text to wrap - even if it means splitting it in the middle of a word:

This paragraph contains a very long word: thisisaveryveryveryveryveryveryverylongword. The long word will break and wrap to the next line.

The CSS code is as follows:

Example

Allow long words to be able to be broken and wrap onto the next line:

```
p {
  word-wrap: break-word;
}
```

CSS Word Breaking

The CSS **word-break** property specifies line breaking rules.

This paragraph contains some text. This line will-break-at-hyphens.

This paragraph co ntains some text. The lines will brea k at any characte r.

The CSS code is as follows:

Example

```
p.test1 {
  word-break: keep-all;
}

p.test2 {
  word-break: break-all;
}
```

CSS Writing Mode

The CSS `writing-mode` property specifies whether lines of text are laid out horizontally or vertically.

Some text with a span element with a vertical-rl writing-mode.

The following example shows some different writing modes:

Example

```
p.test1 {
  writing-mode: horizontal-tb;
}

span.test2 {
  writing-mode: vertical-rl;
}

p.test2 {
  writing-mode: vertical-rl;
}
```

CSS Text Effect Properties

The following table lists the CSS text effect properties:

Property	Description
text-align-last	Specifies how to align the last line of a text
text-justify	Specifies how justified text should be aligned and spaced
text-overflow	Specifies how overflowed content that is not displayed should be signaled to the user
word-break	Specifies line breaking rules for non-CJK scripts
word-wrap	Allows long words to be able to be broken and wrap onto the

	next line
writing-mode	Specifies whether lines of text are laid out horizontally or vertically

CSS Web Fonts

The CSS @font-face Rule

Web fonts allow Web designers to use fonts that are not installed on the user's computer.

When you have found/bought the font you wish to use, just include the font file on your web server, and it will be automatically downloaded to the user when needed.

Your "own" fonts are defined within the CSS **@font-face** rule.

Different Font Formats

TrueType Fonts (TTF)

TrueType is a font standard developed in the late 1980s, by Apple and Microsoft. TrueType is the most common font format for both the Mac OS and Microsoft Windows operating systems.

OpenType Fonts (OTF)

OpenType is a format for scalable computer fonts. It was built on TrueType, and is a registered trademark of Microsoft. OpenType fonts are used commonly today on the major computer platforms.

The Web Open Font Format (WOFF)

WOFF is a font format for use in web pages. It was developed in 2009, and is now a W3C Recommendation. WOFF is essentially OpenType or TrueType with compression and additional metadata. The goal is to support font distribution from a server to a client over a network with bandwidth constraints.

The Web Open Font Format (WOFF 2.0)

TrueType/OpenType font that provides better compression than WOFF 1.0.

SVG Fonts/Shapes

SVG fonts allow SVG to be used as glyphs when displaying text. The SVG 1.1 specification define a font module that allows the creation of fonts within an SVG document. You can also apply CSS to SVG documents, and the @font-face rule can be applied to text in SVG documents.

Embedded OpenType Fonts (EOT)

EOT fonts are a compact form of OpenType fonts designed by Microsoft for use as embedded fonts on web pages.

Browser Support for Font Formats

The numbers in the table specifies the first browser version that fully supports the font format.

Font format					
TTF/OTF	9.0*	4.0	3.5	3.1	10.0
WOFF	9.0	5.0	3.6	5.1	11.1
WOFF2	Not supported	36.0	35.0*	Not supported	26.0
SVG	Not supported	4.0	Not supported	3.2	9.0
EOT	6.0	Not supported	Not supported	Not supported	Not supported

*IE: The font format only works when set to be "installable".

*Firefox: Not supported by default, but can be enabled (need to set a flag to "true" to use WOFF2).

Using The Font You Want

In the **@font-face** rule; first define a name for the font (e.g. myFirstFont) and then point to the font file.

Tip: Always use lowercase letters for the font URL. Uppercase letters can give unexpected results in IE.

To use the font for an HTML element, refer to the name of the font (myFirstFont) through the **font-family** property:

Example

```
@font-face {
  font-family: myFirstFont;
  src: url(sansation_light.woff);
}

div {
  font-family: myFirstFont;
}
```

Using Bold Text

You must add another **@font-face** rule containing descriptors for bold text:

Example

```
@font-face {
  font-family: myFirstFont;
  src: url(sansation_bold.woff);
  font-weight: bold;
}
```

The file "sansation_bold.woff" is another font file, that contains the bold characters for the Sansation font.

Browsers will use this whenever a piece of text with the font-family "myFirstFont" should render as bold.

This way you can have many **@font-face** rules for the same font.

CSS Font Descriptors

The following table lists all the font descriptors that can be defined inside the **@font-face** rule:

Descriptor	Values	Description
font-family	*name*	Required. Defines a name for the font
src	*URL*	Required. Defines the URL of the font file
font-stretch	normal condensed ultra-condensed extra-condensed semi-condensed expanded semi-expanded extra-expanded ultra-expanded	Optional. Defines how the font should be stretched. Default is "normal"
font-style	normal italic oblique	Optional. Defines how the font should be styled. Default is "normal"
font-weight	normal bold 100 200 300 400 500 600 700 800 900	Optional. Defines the boldness of the font. Default is "normal"
unicode-range	*unicode-range*	Optional. Defines the range of UNICODE characters the font supports. Default is "U+0-10FFFF"

CSS 2D Transforms

CSS 2D Transforms

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

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

In this chapter you will learn about the following CSS property:

- **transform**

Browser Support

The numbers in the table specify the first browser version that fully supports the property.

Property					
transform	36.0	10.0	16.0	9.0	23.0

Browser Specific Prefixes

Some older browsers (IE 9) need specific prefixes (-ms-) to understand the 2D transform properties:

Example

```
div {
  -ms-transform: rotate(20deg); /* IE 9 */
  transform: rotate(20deg); /* Standard syntax */
}
```

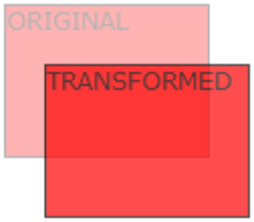
CSS 2D Transforms Methods

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

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

Tip: You will learn about 3D transformations in the next chapter.

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 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 element clockwise with 20 degrees:

Example

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

Using negative values will rotate the element counter-clockwise.

The following example rotates the element counter-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 element to be two times of its original width, and three times of its original height:

Example

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

The following example decreases the element to be half of its original width and height:

Example

```
div {
  transform: scale(0.5, 0.5);
}
```

The scaleX() Method

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

The following example increases the element to be two times of its original width:

Example

```
div {
  transform: scaleX(2);
}
```

The following example decreases the element to be half of its original width:

Example

```
div {
  transform: scaleX(0.5);
}
```

The scaleY() Method

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

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

Example

```
div {
  transform: scaleY(3);
}
```

The following example decreases the element to be half of its original height:

Example

```
div {
  transform: scaleY(0.5);
}
```

The skewX() Method

The **skewX()** method skews an element along the X-axis by the given angle.

The following example skews the element 20 degrees along the X-axis:

Example

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

The skewY() Method

The **skewY()** method skews an element along the Y-axis by the given angle.

The following example skews the element 20 degrees along the Y-axis:

Example

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

The skew() Method

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

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

Example

```
div {
  transform: skew(20deg, 10deg);
}
```

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

Example

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

The matrix() Method

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

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

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

Example

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

CSS Transform Properties

The following table lists all the 2D transform properties:

Property	Description
transform	Applies a 2D or 3D transformation to an element
transform-origin	Allows you to change the position on transformed elements

CSS 2D Transform Methods

Function	Description
matrix(*n*,*n*,*n*,*n*,*n*,*n*)	Defines a 2D transformation, using a matrix of six values
translate(*x*,*y*)	Defines a 2D translation, moving the element along the X- and the Y-axis
translateX(*n*)	Defines a 2D translation, moving the element along the X-axis
translateY(*n*)	Defines a 2D translation, moving the element along the Y-axis
scale(*x*,*y*)	Defines a 2D scale transformation, changing the elements width and height
scaleX(*n*)	Defines a 2D scale transformation, changing the

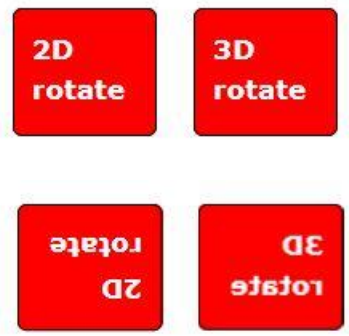
	element's width
scaleY(*n*)	Defines a 2D scale transformation, changing the element's height
rotate(*angle*)	Defines a 2D rotation, the angle is specified in the parameter
skew(*x-angle*,*y-angle*)	Defines a 2D skew transformation along the X- and the Y-axis
skewX(*angle*)	Defines a 2D skew transformation along the X-axis
skewY(*angle*)	Defines a 2D skew transformation along the Y-axis

CSS 3D Transforms

CSS 3D Transforms

CSS also supports 3D transformations.

Mouse over the elements below to see the difference between a 2D and a 3D transformation:



In this chapter you will learn about the following CSS property:

- **transform**

Browser Support

The numbers in the table specify the first browser version that fully supports the property.

Property					
transform	36.0	10.0	16.0	9.0	23.0

CSS 3D Transforms Methods

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);
}
```

CSS Transform Properties

The following table lists all the 3D transform properties:

Property	Description
transform	Applies a 2D or 3D transformation to an element
transform-origin	Allows you to change the position on transformed elements
transform-style	Specifies how nested elements are rendered in 3D space
perspective	Specifies the perspective on how 3D elements are viewed
perspective-origin	Specifies the bottom position of 3D elements
backface-visibility	Defines whether or not an element should be visible when not facing the screen

CSS 3D Transform Methods

Function	Description
matrix3d(*n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n*)	Defines a 3D transformation, using a 4x4 matrix of 16 values
translate3d(*x,y,z*)	Defines a 3D translation
translateX(*x*)	Defines a 3D translation, using only the value for the X-axis
translateY(*y*)	Defines a 3D translation, using only the value for the Y-axis
translateZ(*z*)	Defines a 3D translation, using only the value for the Z-axis
scale3d(*x,y,z*)	Defines a 3D scale transformation
scaleX(*x*)	Defines a 3D scale transformation by giving a value for the X-axis
scaleY(*y*)	Defines a 3D scale transformation by giving a value for the Y-axis
scaleZ(*z*)	Defines a 3D scale transformation by giving a value for the Z-axis
rotate3d(*x,y,z,angle*)	Defines a 3D rotation
rotateX(*angle*)	Defines a 3D rotation along the X-axis
rotateY(*angle*)	Defines a 3D rotation along the Y-axis
rotateZ(*angle*)	Defines a 3D rotation along the Z-axis
perspective(*n*)	Defines a perspective view for a 3D transformed element

CSS Transitions

CSS Transitions

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

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

In this chapter you will learn about the following properties:

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

Browser Support for Transitions

The numbers in the table specify the first browser version that fully supports the property.

Property					
transition	26.0	10.0	16.0	6.1	12.1
transition-delay	26.0	10.0	16.0	6.1	12.1
transition-duration	26.0	10.0	16.0	6.1	12.1
transition-property	26.0	10.0	16.0	6.1	12.1
transition-timing-function	26.0	10.0	16.0	6.1	12.1

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 element. The element has also specified a transition effect for the width property, with a duration of 2 seconds:

Example

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

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

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

Example

```
div:hover {
  width: 300px;
}
```

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

Change Several Property Values

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

Example

```
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:

Example

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

Delay the Transition Effect

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

The following example has a 1 second delay before starting:

Example

```
div {
  transition-delay: 1s;
}
```

Transition + Transformation

The following example adds a transition effect to the transformation:

Example

```
div {
  transition: width 2s, height 2s, transform 2s;
}
```

More Transition Examples

The CSS transition properties can be specified one by one, like this:

Example

```
div {
  transition-property: width;
  transition-duration: 2s;
  transition-timing-function: linear;
  transition-delay: 1s;
}
```

or by using the shorthand property **transition**:

Example

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

CSS Transition Properties

The following table lists all the CSS transition properties:

Property	Description
transition	A shorthand property for setting the four transition properties into a single property
transition-delay	Specifies a delay (in seconds) for the transition effect
transition-duration	Specifies how many seconds or milliseconds a transition effect takes to complete
transition-property	Specifies the name of the CSS property the transition effect is for
transition-timing-function	Specifies the speed curve of the transition effect

CSS Animations

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**

Browser Support for Animations

The numbers in the table specify the first browser version that fully supports the property.

Property	Chrome	Edge	Firefox	Safari	Opera
@keyframes	43.0	10.0	16.0	9.0	30.0
animation-name	43.0	10.0	16.0	9.0	30.0
animation-duration	43.0	10.0	16.0	9.0	30.0
animation-delay	43.0	10.0	16.0	9.0	30.0
animation-iteration-count	43.0	10.0	16.0	9.0	30.0
animation-direction	43.0	10.0	16.0	9.0	30.0
animation-timing-function	43.0	10.0	16.0	9.0	30.0
animation-fill-mode	43.0	10.0	16.0	9.0	30.0
animation	43.0	10.0	16.0	9.0	30.0

Browser Specific Prefixes

Some older browsers need specific prefixes (-webkit-) to understand the animation properties:

Example

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

@keyframes example {
  from {background-color: red;}
  to {background-color: yellow;}
}
```

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

Example

```
/* 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 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;
}
```

The following example will change both the background-color and the position of the 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; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px;
top:0px;}
  50%  {background-color:blue; left:200px;
top:200px;}
  75%  {background-color:green; left:0px;
top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  position: relative;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
}
```

Delay an Animation

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

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

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;
}
```

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

Example

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

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

Example

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

The following example lets the element get the style values set by the first keyframe before the animation starts (during the animation-delay period):

Example

```
div {
  width: 100px;
  height: 100px;
  background: red;
  position: relative;
  animation-name: example;
  animation-duration: 3s;
  animation-delay: 2s;
  animation-fill-mode: backwards;
}
```

The following example lets the element get the style values set by the first keyframe before the animation starts, and 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-delay: 2s;
  animation-fill-mode: both;
}
```

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;
}
```

CSS Animation Properties

The following table lists the @keyframes rule and all the CSS animation properties:

Property	Description
@keyframes	Specifies the animation code
animation	A shorthand property for setting all the animation properties
animation-delay	Specifies a delay for the start

	of an animation
animation-direction	Specifies whether an animation should be played forwards, backwards or in alternate cycles
animation-duration	Specifies how long time an animation should take to complete one cycle
animation-fill-mode	Specifies a style for the element when the animation is not playing (before it starts, after it ends, or both)
animation-iteration-count	Specifies the number of times an animation should be played
animation-name	Specifies the name of the @keyframes animation
animation-play-state	Specifies whether the animation is running or paused
animation-timing-function	Specifies the speed curve of the animation

CSS Tooltip

Create tooltips with CSS.

Demo: Tooltip Examples

A tooltip is often used to specify extra information about something when the user moves the mouse pointer over an element:

Basic Tooltip

Create a tooltip that appears when the user moves the mouse over an element:

Example

```
<style>
/* Tooltip container */
.tooltip {
  position: relative;
  display: inline-block;
  border-bottom: 1px dotted black; /* If you want
dots under the hoverable text */
}

/* Tooltip text */
.tooltip .tooltiptext {
  visibility: hidden;
  width: 120px;
  background-color: black;
  color: #fff;
  text-align: center;
  padding: 5px 0;
  border-radius: 6px;

  /* Position the tooltip text - see examples
below! */
  position: absolute;
  z-index: 1;
}

/* Show the tooltip text when you mouse over the
tooltip container */
.tooltip:hover .tooltiptext {
  visibility: visible;
}
</style>

<div class="tooltip">Hover over me
  <span class="tooltiptext">Tooltip text</span>
</div>
```

Example Explained

HTML: Use a container element (like) and add the **"tooltip"** class to it. When the user mouse over this , it will show the tooltip text.

The tooltip text is placed inside an inline element (like) with **class="tooltiptext"**.

CSS: The **tooltip** class use **position:relative**, which is needed to position the tooltip text (**position:absolute**). **Note:** See examples below on how to position the tooltip.

The **tooltiptext** class holds the actual tooltip text. It is hidden by default, and will be visible on hover (see below). We have also added some basic styles to it: 120px width, black

background color, white text color, centered text, and 5px top and bottom padding.

The CSS **border-radius** property is used to add rounded corners to the tooltip text.

The **:hover** selector is used to show the tooltip text when the user moves the mouse over the with **class="tooltip"**.

Positioning Tooltips

In this example, the tooltip is placed to the right (**left:105%**) of the "hoverable" text ( ). Also note that **top:-5px** is used to place it in the middle of its container element. We use the number **5** because the tooltip text has a top and bottom padding of 5px. If you increase its padding, also increase the value of the **top** property to ensure that it stays in the middle (if this is something you want). The same applies if you want the tooltip placed to the left.

Right Tooltip

```
.tooltip .tooltiptext {
  top: -5px;
  left: 105%;
}
```

Result:

Left Tooltip

```
.tooltip .tooltiptext {
  top: -5px;
  right: 105%;
}
```

Result:

If you want the tooltip to appear on top or on the bottom, see examples below. Note that we use the **margin-left** property with a value of minus 60 pixels. This is to center the tooltip above/below the hoverable text. It is set to the half of the tooltip's width (120/2 = 60).

Top Tooltip

```
.tooltip .tooltiptext {
  width: 120px;
  bottom: 100%;
  left: 50%;
  margin-left: -60px; /* Use half of the width
(120/2 = 60), to center the tooltip */
}
```

Result:

Bottom Tooltip

```
.tooltip .tooltiptext {
  width: 120px;
  top: 100%;
  left: 50%;
  margin-left: -60px; /* Use half of the width
(120/2 = 60), to center the tooltip */
}
```

Result:

Tooltip Arrows

To create an arrow that should appear from a specific side of the tooltip, add "empty" content after tooltip, with the pseudo-element class **::after** together with the **content** property. The arrow itself is created using borders. This will make the tooltip look like a speech bubble.

This example demonstrates how to add an arrow to the bottom of the tooltip:

Bottom Arrow

```
.tooltip .tooltiptext::after {
  content: " ";
  position: absolute;
  top: 100%; /* At the bottom of the tooltip */
  left: 50%;
  margin-left: -5px;
  border-width: 5px;
  border-style: solid;
  border-color: black transparent transparent
transparent;
}
```

Result:

Hover over me

Example Explained

Position the arrow inside the tooltip: **top: 100%** will place the arrow at the bottom of the tooltip. **left: 50%** will center the arrow.

Note: The **border-width** property specifies the size of the arrow. If you change this, also change the **margin-left** value to the same. This will keep the arrow centered.

The **border-color** is used to transform the content into an arrow. We set the top border to black, and the rest to transparent. If all sides were black, you would end up with a black square box.

This example demonstrates how to add an arrow to the top of the tooltip. Notice that we set the bottom border color this time:

Top Arrow

```
.tooltip .tooltiptext::after {
  content: " ";
  position: absolute;
  bottom: 100%;  /* At the top of the tooltip */
  left: 50%;
  margin-left: -5px;
  border-width: 5px;
  border-style: solid;
  border-color: transparent transparent black
transparent;
}
```

Result:

Hover over me

This example demonstrates how to add an arrow to the left of the tooltip:

Left Arrow

```
.tooltip .tooltiptext::after {
  content: " ";
  position: absolute;
  top: 50%;
  right: 100%; /* To the left of the tooltip */
  margin-top: -5px;
  border-width: 5px;
  border-style: solid;
  border-color: transparent black transparent
transparent;
}
```

Result:

Hover over me

This example demonstrates how to add an arrow to the right of the tooltip:

Right Arrow

```
.tooltip .tooltiptext::after {
  content: " ";
  position: absolute;
  top: 50%;
  left: 100%; /* To the right of the tooltip */
  margin-top: -5px;
  border-width: 5px;
  border-style: solid;
  border-color: transparent transparent
transparent black;
}
```

Result:

Hover over me

Fade In Tooltips (Animation)

If you want to fade in the tooltip text when it is about to be visible, you can use the CSS **transition** property together with the **opacity** property, and go from being completely invisible

to 100% visible, in a number of specified seconds (1 second in our example):

Example

```
.tooltip .tooltiptext {
  opacity: 0;
  transition: opacity 1s;
}

.tooltip:hover .tooltiptext {
  opacity: 1;
}
```

CSS Styling Images

Learn how to style images using CSS.

Rounded Images

Use the **border-radius** property to create rounded images:

Example

Rounded Image:

```
img {
  border-radius: 8px;
}
```

Example

Circled Image:

```
img {
  border-radius: 50%;
}
```

Thumbnail Images

Use the **border** property to create thumbnail images.

Thumbnail Image:

Thumbnail Image:

Example

```
img {
  border: 1px solid #ddd;
  border-radius: 4px;
  padding: 5px;
  width: 150px;
}

<img src="paris.jpg" alt="Paris">
```

Thumbnail Image as Link:

Example

```
img {
  border: 1px solid #ddd;
  border-radius: 4px;
  padding: 5px;
  width: 150px;
}

img:hover {
```

Responsive Images

Responsive images will automatically adjust to fit the size of the screen.

Resize the browser window to see the effect:

If you want an image to scale down if it has to, but never scale up to be larger than its original size, add the following:

Example

```
img {
  max-width: 100%;
  height: auto;
}
```

Tip: Read more about Responsive Web Design in our CSS RWD Tutorial.

Center an Image

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

Example

```
img {
  display: block;
  margin-left: auto;
  margin-right: auto;
  width: 50%;
}
```

Polaroid Images / Cards

Northern Lights

Example

```
div.polaroid {
  width: 80%;
  background-color: white;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0
6px 20px 0 rgba(0, 0, 0, 0.19);
}

img {width: 100%}

div.container {
  text-align: center;
  padding: 10px 20px;
}
```

Transparent Image

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

opacity 0.2

opacity 0.5

opacity 1
(default)

Example

```
img {
  opacity: 0.5;
}
```

Image Text

How to position text in an image:

Example

Try it Yourself:

Image Filters

The CSS **filter** property adds visual effects (like blur and saturation) to an element.

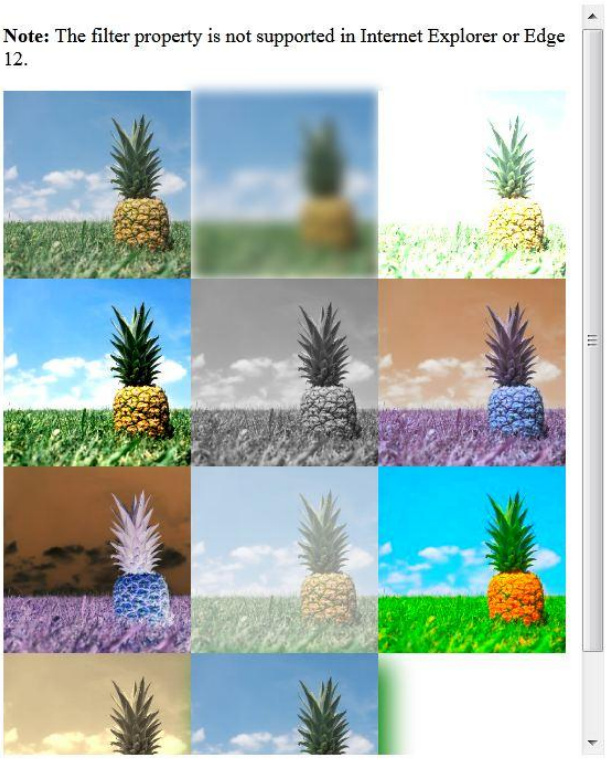
Note: The filter property is not supported in Internet Explorer or Edge 12.

Example

Change the color of all images to black and white (100% gray):

```
img {
  filter: grayscale(100%);
}
```

Note: The filter property is not supported in Internet Explorer or Edge 12.



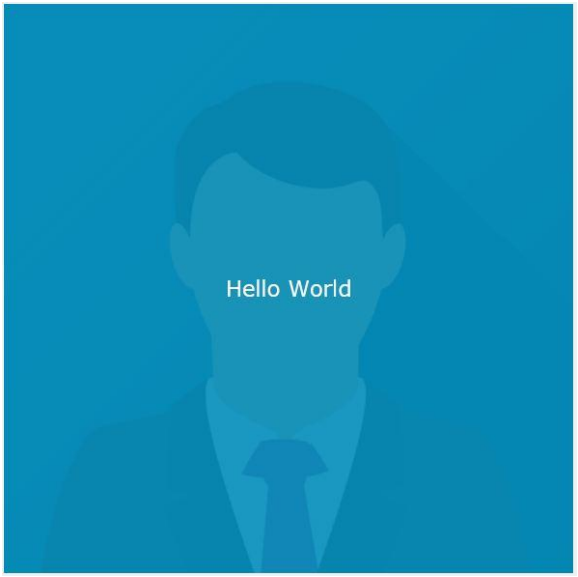
Tip: Go to our CSS filter Reference to learn more about CSS filters.

Image Hover Overlay

Create an overlay effect on hover:

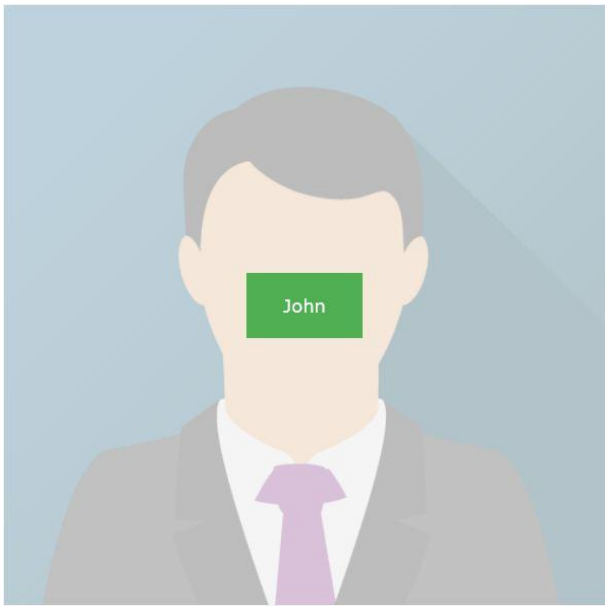
Example

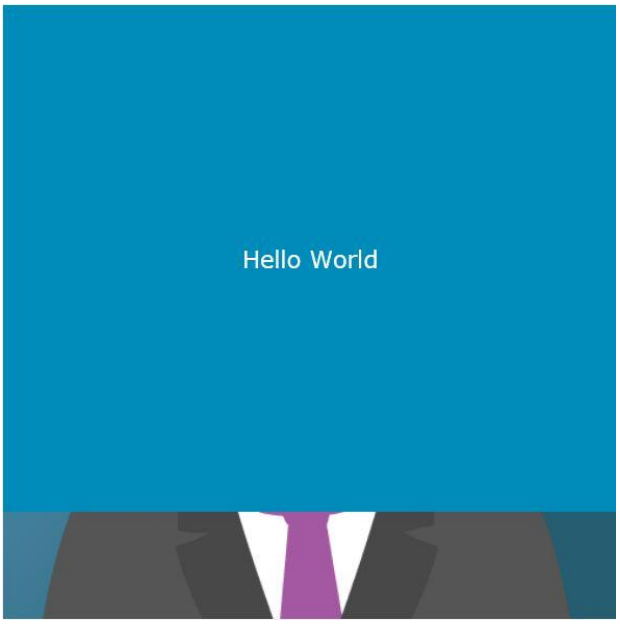
Fade in text:



Example

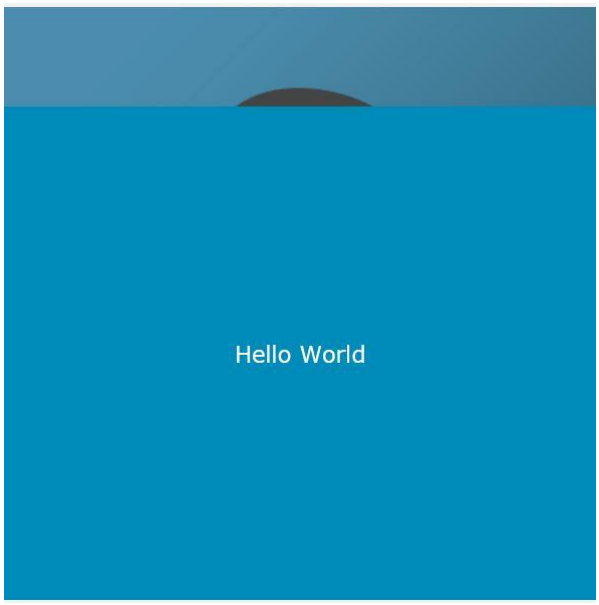
Fade in a box:





Example

Slide in (bottom):



Example

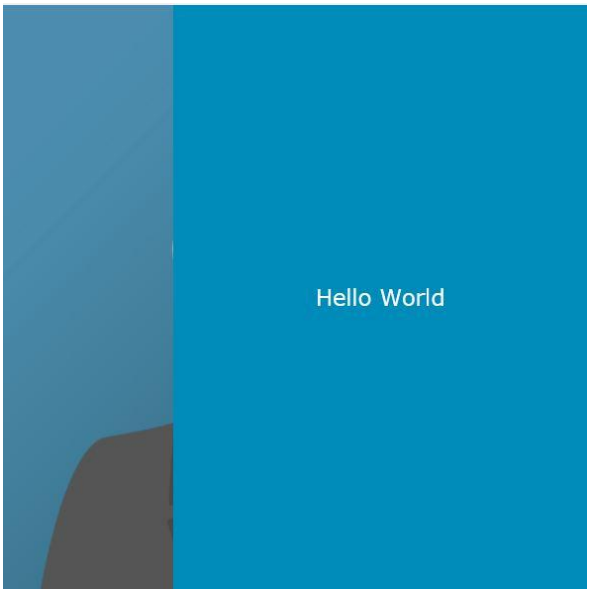
Slide in (top):

Example

Slide in (left):

Example

Slide in (right):



Flip an Image

Move your mouse over the image:

Example

```
img:hover {
  transform: scaleX(-1);
}
```

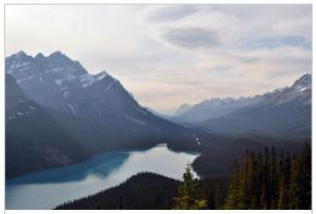
Responsive Image Gallery

CSS can be used to create image galleries. This example use media queries to re-arrange the images on different screen sizes. Resize the browser window to see the effect:

Add a description of the image here

Add a description of the image here

Add a description of the image here



Add a description of the image here

Example

```
.responsive {
  padding: 0 6px;
  float: left;
  width: 24.99999%;
}

@media only screen and (max-width: 700px){
  .responsive {
    width: 49.99999%;
    margin: 6px 0;
  }
}

@media only screen and (max-width: 500px){
  .responsive {
    width: 100%;
  }
}
```

Tip: Read more about Responsive Web Design in our CSS RWD Tutorial.

Image Modal (Advanced)

This is an example to demonstrate how CSS and JavaScript can work together.

First, use CSS to create a modal window (dialog box), and hide it by default.

Then, use a JavaScript to show the modal window and to display the image inside the modal, when a user clicks on the image:

Example

```
// Get the modal
var modal = document.getElementById('myModal');

// Get the image and insert it inside the modal -
use its "alt" text as a caption
var img = document.getElementById('myImg');
var modalImg = document.getElementById("img01");
var captionText =
document.getElementById("caption");
img.onclick = function(){
  modal.style.display = "block";
  modalImg.src = this.src;
  captionText.innerHTML = this.alt;
}

// Get the <span> element that closes the modal
var span =
document.getElementsByClassName("close")[0];

// When the user clicks on <span> (x), close the
modal
span.onclick = function() {
  modal.style.display = "none";
}
```

CSS The object-fit Property

The CSS **object-fit** property is used to specify how an <img> or <video> should be resized to fit its container.

Browser Support

The numbers in the table specify the first browser version that fully supports the property.

Property					
object-fit	31.0	16.0	36.0	7.1	19.0

The CSS object-fit Property

The CSS **object-fit** property is used to specify how an <img> or <video> should be resized to fit its container.

This property tells the content to fill the container in a variety of ways; such as "preserve that aspect ratio" or "stretch up and take up as much space as possible".

Look at the following image from Paris, which is 400x300 pixels:

However, if we style the image above to be 200x400 pixels, it will look like this:

Example

```
img {
  width: 200px;
  height: 400px;
}
```

We see that the image is being squeezed to fit the container of 200x400 pixels, and its original aspect ratio is destroyed.

If we use **object-fit: cover**; it will cut off the sides of the image, preserving the aspect ratio, and also filling in the space, like this:

Example

```
img {
  width: 200px;
  height: 400px;
  object-fit: cover;
}
```

Another Example

Here we have two images and we want them to fill the width of 50% of the browser window and 100% of the height.

In the following example we do NOT use **object-fit**, so when we resize the browser window, the aspect ratio of the images will be destroyed:

Example

In the next example, we use **object-fit: cover**;, so when we resize the browser window, the aspect ratio of the images is preserved:

Example

All Values of The CSS object-fit Property

The **object-fit** property can have the following values:

- **fill** - This is default. The replaced content is sized to fill the element's content box. If necessary, the object will be stretched or squished to fit
- **contain** - The replaced content is scaled to maintain its aspect ratio while fitting within the element's content box
- **cover** - The replaced content is sized to maintain its aspect ratio while filling the element's entire content box. The object will be clipped to fit
- **none** - The replaced content is not resized
- **scale-down** - The content is sized as if none or contain were specified (would result in a smaller concrete object size)

The following example demonstrates all the possible values of the **object-fit** property:

Example

```
.fill {object-fit: fill;}
.contain {object-fit: contain;}
.cover {object-fit: cover;}
.scale-down {object-fit: scale-down;}
.none {object-fit: none;}
```

CSS Buttons

Learn how to style buttons using CSS.

Basic Button Styling

Example

```
.button {
  background-color: #4CAF50; /* Green */
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
}
```

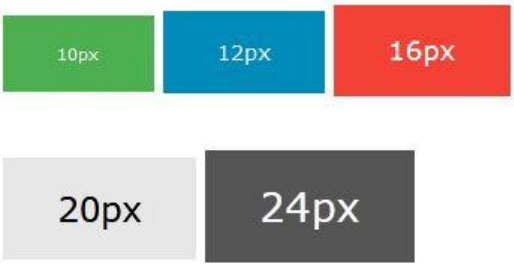
Button Colors

Use the **background-color** property to change the background color of a button:

Example

```
.button1 {background-color: #4CAF50;} /* Green */
.button2 {background-color: #008CBA;} /* Blue */
.button3 {background-color: #f44336;} /* Red */
.button4 {background-color: #e7e7e7; color:
black;} /* Gray */
.button5 {background-color: #555555;} /* Black */
```

Button Sizes



Use the **font-size** property to change the font size of a button:

Example

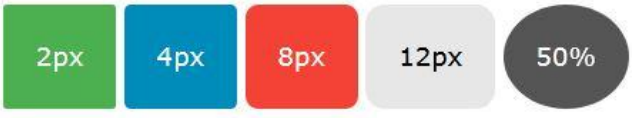
```
.button1 {font-size: 10px;}
.button2 {font-size: 12px;}
.button3 {font-size: 16px;}
.button4 {font-size: 20px;}
.button5 {font-size: 24px;}
```

Use the **padding** property to change the padding of a button:

Example

```
.button1 {padding: 10px 24px;}
.button2 {padding: 12px 28px;}
.button3 {padding: 14px 40px;}
.button4 {padding: 32px 16px;}
.button5 {padding: 16px;}
```

Rounded Buttons

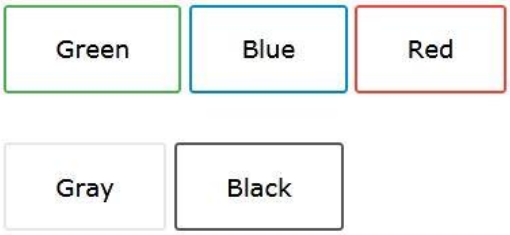


Use the **border-radius** property to add rounded corners to a button:

Example

```
.button1 {border-radius: 2px;}
.button2 {border-radius: 4px;}
.button3 {border-radius: 8px;}
.button4 {border-radius: 12px;}
.button5 {border-radius: 50%;}
```

Colored Button Borders

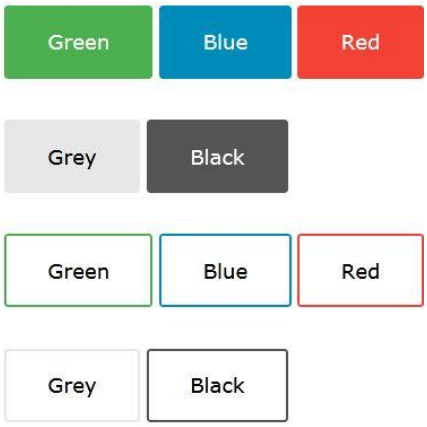


Use the **border** property to add a colored border to a button:

Example

```
.button1 {
  background-color: white;
  color: black;
  border: 2px solid #4CAF50; /* Green */
}
...
```

Hoverable Buttons



Use the **:hover** selector to change the style of a button when you move the mouse over it.

Tip: Use the **transition-duration** property to determine the speed of the "hover" effect:

Example

```
.button {
  transition-duration: 0.4s;
}

.button:hover {
  background-color: #4CAF50; /* Green */
  color: white;
}
...
```

Shadow Buttons

Use the **box-shadow** property to add shadows to a button:

Example

```
.button1 {
  box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2), 0 6px
20px 0 rgba(0,0,0,0.19);
}

.button2:hover {
  box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0
17px 50px 0 rgba(0,0,0,0.19);
}
```

Disabled Buttons

Use the **opacity** property to add transparency to a button (creates a "disabled" look).

Tip: You can also add the **cursor** property with a value of "not-allowed", which will display a "no parking sign" when you mouse over the button:

Example

```
.disabled {
  opacity: 0.6;
  cursor: not-allowed;
}
```

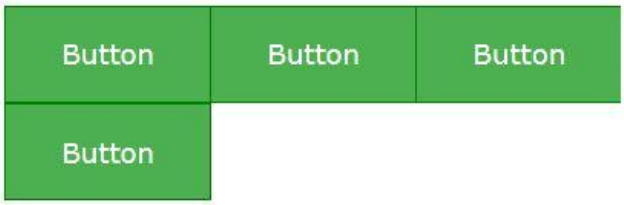
Button Width

By default, the size of the button is determined by its text content (as wide as its content). Use the **width** property to change the width of a button:

Example

```
.button1 {width: 250px;}
.button2 {width: 50%;}
.button3 {width: 100%;}
```

Button Groups

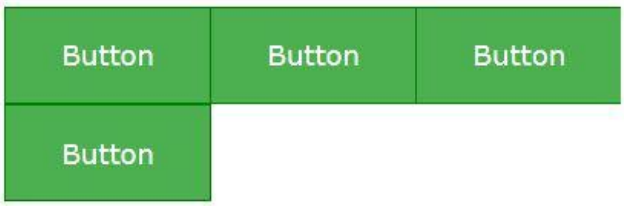


Remove margins and add **float:left** to each button to create a button group:

Example

```
.button {
  float: left;
}
```

Bordered Button Group



Use the **border** property to create a bordered button group:

Example

```
.button {
  float: left;
  border: 1px solid green;
}
```

Vertical Button Group

Use **display:block** instead of **float:left** to group the buttons below each other, instead of side by side:

Example

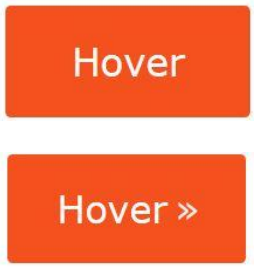
```
.button {
  display: block;
}
```

Button on Image

Animated Buttons

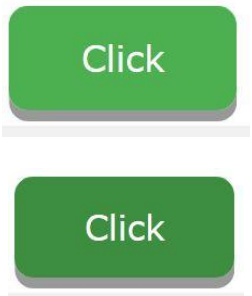
Example

Add an arrow on hover:



Example

Add a "pressed" effect on click:

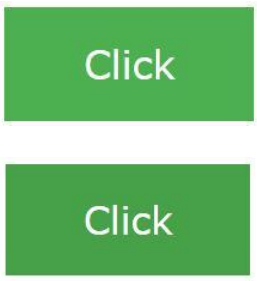


Example

Fade in on hover:

Example

Add a "ripple" effect on click:



CSS Pagination Examples

Learn how to create a responsive pagination using CSS.

Simple Pagination

If you have a website with lots of pages, you may wish to add some sort of pagination to each page:

Example

```
.pagination {
  display: inline-block;
}

.pagination a {
  color: black;
  float: left;
  padding: 8px 16px;
  text-decoration: none;
}
```

Active and Hoverable Pagination

Highlight the current page with an **.active** class, and use the **:hover** selector to change the color of each page link when moving the mouse over them:

Example

```
.pagination a.active {
  background-color: #4CAF50;
  color: white;
}

.pagination a:hover:not(.active) {background-color: #ddd;}
```

Rounded Active and Hoverable Buttons

Add the **border-radius** property if you want a rounded "active" and "hover" button:

Example

```
.pagination a {
  border-radius: 5px;
}

.pagination a.active {
  border-radius: 5px;
}
```

Hoverable Transition Effect

Example

```
.pagination a {
  transition: background-color .3s;
}
```

Bordered Pagination

Use the **border** property to add borders to the pagination:

Example

```
.pagination a {
  border: 1px solid #ddd; /* Gray */
}
```

Rounded Borders

Tip: Add rounded borders to your first and last link in the pagination:

Example

```
.pagination a:first-child {
  border-top-left-radius: 5px;
  border-bottom-left-radius: 5px;
}

.pagination a:last-child {
  border-top-right-radius: 5px;
  border-bottom-right-radius: 5px;
}
```

Space Between Links

Tip: Add the **margin** property if you do not want to group the page links:

Example

```
.pagination a {
  margin: 0 4px; /* 0 is for top and bottom. Feel
free to change it */
}
```

Pagination Size

Change the size of the pagination with the **font-size** property:

Example

```
.pagination a {
  font-size: 22px;
}
```

Centered Pagination

Example

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

More Examples

Example

Next/Previous buttons:

Navigation pagination:

Breadcrumbs

Another variation of pagination is so-called "breadcrumbs":

Example

```
ul.breadcrumb {
  padding: 8px 16px;
  list-style: none;
  background-color: #eee;
}

ul.breadcrumb li {display: inline;}

ul.breadcrumb li+li:before {
  padding: 8px;
  color: black;
  content: "/\00a0";
}
```

CSS Multiple Columns

CSS Multi-column Layout

The CSS multi-column layout allows easy definition of multiple columns of text - just like in newspapers:

CSS Multi-column Properties

In this chapter you will learn about the following multi-column properties:

- **column-count**
- **column-gap**
- **column-rule-style**
- **column-rule-width**
- **column-rule-color**
- **column-rule**
- **column-span**
- **column-width**

Browser Support

The numbers in the table specify the first browser version that fully supports the property.

Property					
column-count	50.0	10.0	52.0	9.0	37.0
column-gap	50.0	10.0	52.0	9.0	37.0
column-rule	50.0	10.0	52.0	9.0	37.0
column-rule-color	50.0	10.0	52.0	9.0	37.0
column-rule-style	50.0	10.0	52.0	9.0	37.0
column-rule-width	50.0	10.0	52.0	9.0	37.0
column-span	50.0	10.0	71.0	9.0	37.0
column-width	50.0	10.0	52.0	9.0	37.0

CSS Create Multiple Columns

The **column-count** property specifies the number of columns an element should be divided into.

The following example will divide the text in the element into 3 columns:

Example

```
div {
  column-count: 3;
}
```

CSS Column Rules

The **column-rule-style** property specifies the style of the rule between columns:

Example

```
div {
  column-rule-style: solid;
}
```

The **column-rule-width** property specifies the width of the rule between columns:

Example

```
div {
  column-rule-width: 1px;
}
```

The **column-rule-color** property specifies the color of the rule between columns:

Example

```
div {
  column-rule-color: lightblue;
}
```

The **column-rule** property is a shorthand property for setting all the column-rule-* properties above.

The following example sets the width, style, and color of the rule between columns:

Example

```
div {
  column-rule: 1px solid lightblue;
}
```

Specify How Many Columns an Element Should Span

The **column-span** property specifies how many columns an element should span across.

The following example specifies that the <h2> element should span across all columns:

Example

```
h2 {
  column-span: all;
}
```

Specify The Column Width

The **column-width** property specifies a suggested, optimal width for the columns.

The following example specifies that the suggested, optimal width for the columns should be 100px:

Example

```
div {
  column-width: 100px;
}
```

CSS Multi-columns Properties

The following table lists all the multi-columns properties:

Property	Description
column-count	Specifies the number of columns an element should be divided into
column-fill	Specifies how to fill columns
column-gap	Specifies the gap between the columns
column-rule	A shorthand property for setting all the column-rule-* properties
column-rule-color	Specifies the color of the rule between columns
column-rule-style	Specifies the style of the rule between columns
column-rule-width	Specifies the width of the rule between columns
column-span	Specifies how many columns an element should span across
column-width	Specifies a suggested, optimal width for the columns
columns	A shorthand property for setting column-width and column-count

CSS User Interface

CSS User Interface

In this chapter you will learn about the following CSS user interface properties:

- **resize**
- **outline-offset**

Browser Support

The numbers in the table specify the first browser version that fully supports the property.

Property					
resize	4.0	Not supported	5.0	4.0	15.0
outline-offset	4.0	15.0	5.0	4.0	9.5

CSS Resizing

The **resize** property specifies if (and how) an element should be resizable by the user.

This div element is resizable by the user!

To resize: Click and drag the bottom right corner of this div element.

Note: Internet Explorer does not support the resize property.

The following example lets the user resize only the width of a element:

Example

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

The following example lets the user resize only the height of a element:

Example

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

The following example lets the user resize both the height and width of a element:

Example

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

In many browsers, <textarea> is resizable by default. Here, we have used the resize property to disable the resizability:

```
textarea {
  resize: none;
}
```

CSS Outline Offset

The **outline-offset** property adds space between an outline and the edge or border of an element.

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

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.

The following example uses the outline-offset property to add space between the border and the outline:

Example

```
div.ex1 {
  margin: 20px;
  border: 1px solid black;
  outline: 4px solid red;
  outline-offset: 15px;
}

div.ex2 {
  margin: 10px;
  border: 1px solid black;
  outline: 5px dashed blue;
  outline-offset: 5px;
}
```

CSS User Interface Properties

The following table lists all the user interface properties:

Property	Description
outline-offset	Adds space between an outline and the edge or border of an element
resize	Specifies whether or not an element is resizable by the user

CSS Variables

CSS Custom Properties (Variables)

The **var()** function can be used to insert the value of a custom property.

Browser Support

The numbers in the table specify the first browser version that fully supports the property.

Function	Chrome	Edge	Firefox	Safari	Opera
var()	49.0	15.0	31.0	9.1	36.0

The var() Function

Variables in CSS should be declared within a CSS selector that defines its scope. For a global scope you can use either the :root or the body selector.

The variable name must begin with two dashes (--) and is case sensitive!

The syntax of the **var()** function is as follows:

```
var(custom-name, value)
```

Value	Description
custom-name	Required. The custom property's name (must start with two dashes)
value	Optional. The fallback value (used if the custom property is invalid)

The following example first defines a global custom property named "--main-bg-color", then it uses the var() function to insert the value of the custom property later in the style sheet:

Example

```
:root {
  --main-bg-color: coral;
}

#div1 {
  background-color: var(--main-bg-color);
}

#div2 {
  background-color: var(--main-bg-color);
}
```

The following example uses the var() function to insert several custom property values:

Example

```
:root {
  --main-bg-color: coral;
  --main-txt-color: blue;
  --main-padding: 15px;
}

#div1 {
  background-color: var(--main-bg-color);
  color: var(--main-txt-color);
  padding: var(--main-padding);
}

#div2 {
  background-color: var(--main-bg-color);
  color: var(--main-txt-color);
  padding: var(--main-padding);
}
```

CSS var() Function

Property	Description
var()	Inserts the value of a custom property

CSS Box Sizing

CSS Box Sizing

The CSS **box-sizing** property allows us to include the padding and border in an element's total width and height.

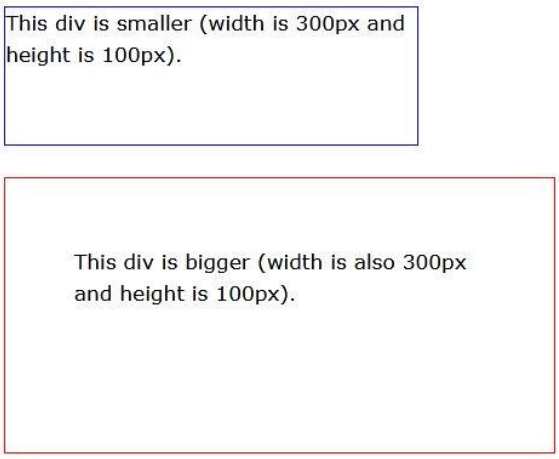
Without the CSS box-sizing Property

By default, the width and height of an element is calculated like this:

width + padding + border = actual width of an element
height + padding + border = actual height of an element

This means: When you set the width/height of an element, the element often appears bigger than you have set (because the element's border and padding are added to the element's specified width/height).

The following illustration shows two elements with the same specified width and height:



The two elements above end up with different sizes in the result (because div2 has a padding specified):

Example

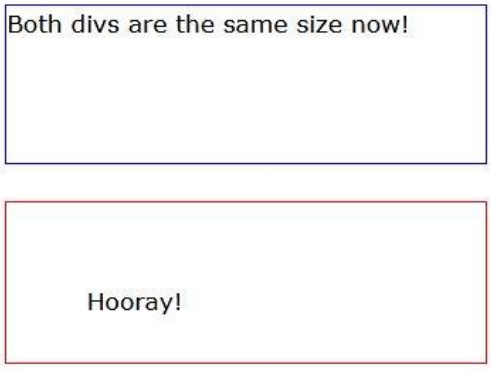
```
.div1 {
  width: 300px;
  height: 100px;
  border: 1px solid blue;
}

.div2 {
  width: 300px;
  height: 100px;
  padding: 50px;
  border: 1px solid red;
}
```

The **box-sizing** property solves this problem.

With the CSS box-sizing Property

The **box-sizing** property allows us to include the padding and border in an element's total width and height.

If you set **box-sizing: border-box;** on an element, padding and border are included in the width and height:



Here is the same example as above, with **box-sizing: border-box;** added to both elements:

```
.div1 {
  width: 300px;
  height: 100px;
  border: 1px solid blue;
  box-sizing: border-box;
}

.div2 {
  width: 300px;
  height: 100px;
  padding: 50px;
  border: 1px solid red;
  box-sizing: border-box;
}
```

Since the result of using the **box-sizing: border-box;** is so much better, many developers want all elements on their pages to work this way.

The code below ensures that all elements are sized in this more intuitive way. Many browsers already use **box-sizing: border-box;** for many form elements (but not all - which is why inputs and text areas look different at width: 100%;).

Applying this to all elements is safe and wise:

Example

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

CSS Box Sizing Property

Property	Description
box-sizing	Defines how the width and height of an element are calculated: should they include padding and borders, or not

CSS Flexbox

CSS Flexbox Layout Module

Before the Flexbox Layout module, there were four layout modes:

- Block, for sections in a webpage
- Inline, for text
- Table, for two-dimensional table data
- Positioned, for explicit position of an element

The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.

Browser Support

The flexbox properties are supported in all modern browsers.

29.0	11.0	22.0	10	48

Flexbox Elements

To start using the Flexbox model, you need to first define a flex container.

The element above represents a flex container (the blue area) with three flex items.

Example

A flex container with three flex items:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

Parent Element (Container)

The flex container becomes flexible by setting the **display** property to *flex*:

Example

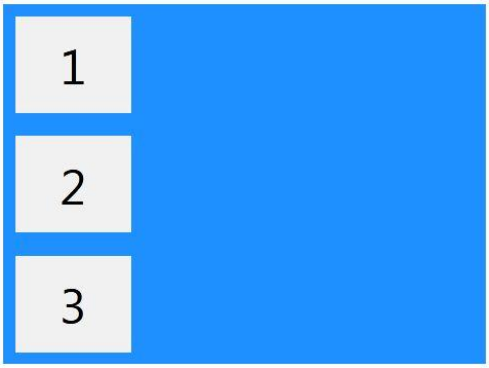
```
.flex-container {
  display: flex;
}
```

The flex container properties are:

- flex-direction
- flex-wrap
- flex-flow
- justify-content
- align-items
- align-content

The flex-direction Property

The **flex-direction** property defines in which direction the container wants to stack the flex items.



Example

The *column* value stacks the flex items vertically (from top to bottom):

```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

Example

The *column-reverse* value stacks the flex items vertically (but from bottom to top):

```
.flex-container {
  display: flex;
  flex-direction: column-reverse;
}
```

Example

The *row* value stacks the flex items horizontally (from left to right):

```
.flex-container {
  display: flex;
  flex-direction: row;
}
```

Example

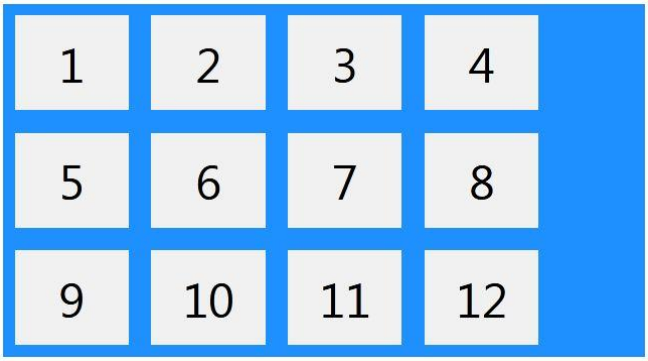
The *row-reverse* value stacks the flex items horizontally (but from right to left):

```
.flex-container {
  display: flex;
  flex-direction: row-reverse;
}
```

The flex-wrap Property

The **flex-wrap** property specifies whether the flex items should wrap or not.

The examples below have 12 flex items, to better demonstrate the **flex-wrap** property.



Example

The *wrap* value specifies that the flex items will wrap if necessary:

```
.flex-container {
  display: flex;
  flex-wrap: wrap;
}
```

Example

The *nowrap* value specifies that the flex items will not wrap (this is default):

```
.flex-container {
  display: flex;
  flex-wrap: nowrap;
}
```

Example

The *wrap-reverse* value specifies that the flexible items will wrap if necessary, in reverse order:

```
.flex-container {
  display: flex;
  flex-wrap: wrap-reverse;
}
```

The flex-flow Property

The **flex-flow** property is a shorthand property for setting both the **flex-direction** and **flex-wrap properties.**

```
.flex-container {
  display: flex;
  flex-flow: row wrap;
}
```

The justify-content Property

The **justify-content** property is used to align the flex items:

Example

The *center* value aligns the flex items at the center of the container:

```
.flex-container {
  display: flex;
  justify-content: center;
}
```

Example

The *flex-start* value aligns the flex items at the beginning of the container (this is default):

```
.flex-container {
  display: flex;
  justify-content: flex-start;
}
```

Example

The *flex-end* value aligns the flex items at the end of the container:

```
.flex-container {
  display: flex;
  justify-content: flex-end;
}
```

Example

The *space-around* value displays the flex items with space before, between, and after the lines:

```
.flex-container {
  display: flex;
  justify-content: space-around;
}
```

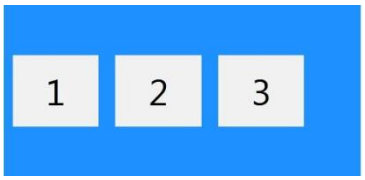
Example

The *space-between* value displays the flex items with space between the lines:

```
.flex-container {
  display: flex;
  justify-content: space-between;
}
```

The align-items Property

The **align-items** property is used to align the flex items vertically.



In these examples we use a 200 pixels high container, to better demonstrate the **align-items** property.

Example

The *center* value aligns the flex items in the middle of the container:

```
.flex-container {
  display: flex;
  height: 200px;
  align-items: center;
}
```

Example

The *flex-start* value aligns the flex items at the top of the container:

```
.flex-container {
  display: flex;
  height: 200px;
  align-items: flex-start;
}
```

Example

The *flex-end* value aligns the flex items at the bottom of the container:

```
.flex-container {
  display: flex;
  height: 200px;
  align-items: flex-end;
}
```

Example

The *stretch* value stretches the flex items to fill the container (this is default):

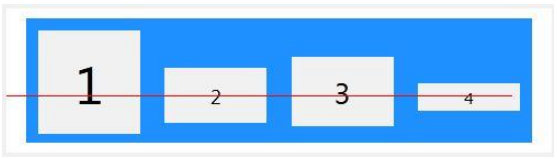
```
.flex-container {
  display: flex;
  height: 200px;
  align-items: stretch;
}
```

Example

The *baseline* value aligns the flex items such as their baselines aligns:

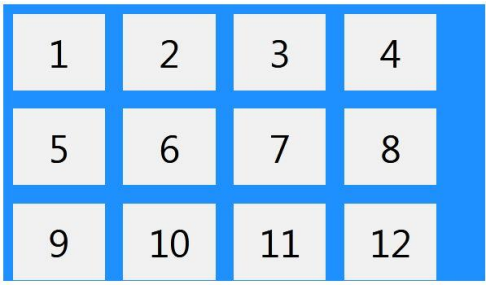
```
.flex-container {
  display: flex;
  height: 200px;
  align-items: baseline;
}
```

Note: the example uses different font-size to demonstrate that the items gets aligned by the text baseline:



The align-content Property

The **align-content** property is used to align the flex lines.



In these examples we use a 600 pixels high container, with the flex-wrap property set to *wrap*, to better demonstrate the **align-content** property.

Example

The *space-between* value displays the flex lines with equal space between them:

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: space-between;
}
```

Example

The *space-around* value displays the flex lines with space before, between, and after them:

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: space-around;
}
```

Example

The *stretch* value stretches the flex lines to take up the remaining space (this is default):

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: stretch;
}
```

Example

The *center* value displays display the flex lines in the middle of the container:

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: center;
}
```

Example

The *flex-start* value displays the flex lines at the start of the container:

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: flex-start;
}
```

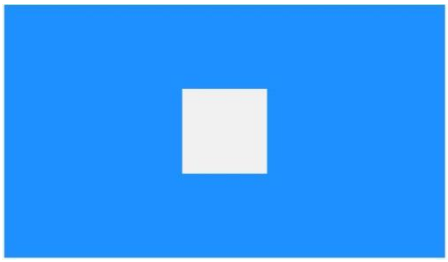
Example

The *flex-end* value displays the flex lines at the end of the container:

```
.flex-container {
  display: flex;
  height: 600px;
  flex-wrap: wrap;
  align-content: flex-end;
}
```

Perfect Centering

In the following example we will solve a very common style problem: perfect centering.



SOLUTION: Set both the **justify-content** and **align-items** properties to *center*, and the flex item will be perfectly centered:

Example

```
.flex-container {
  display: flex;
  height: 300px;
  justify-content: center;
  align-items: center;
}
```

Child Elements (Items)

The direct child elements of a flex container automatically becomes flexible (flex) items.

The element above represents four blue flex items inside a grey flex container.

Example

```
<div class="flex-container">
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
  <div style="order: 1">4</div>
</div>
```

The flex item properties are:

- order
- flex-grow
- flex-shrink
- flex-basis
- flex
- align-self

The order Property

The **order** property specifies the order of the flex items.

The first flex item in the code does not have to appear as the first item in the layout.

The order value must be a number, default value is 0.

Example

The *order* property can change the order of the flex items:

```
<div class="flex-container">
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
  <div style="order: 1">4</div>
</div>
```

The flex-grow Property

The **flex-grow** property specifies how much a flex item will grow relative to the rest of the flex items.

The value must be a number, default value is 0.

Example

Make the third flex item grow eight times faster than the other flex items:

```
<div class="flex-container">
  <div style="flex-grow: 1">1</div>
  <div style="flex-grow: 1">2</div>
  <div style="flex-grow: 8">3</div>
</div>
```

The flex-shrink Property

The **flex-shrink** property specifies how much a flex item will shrink relative to the rest of the flex items.

The value must be a number, default value is 1.

Example

Do not let the third flex item shrink as much as the other flex items:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-shrink: 0">3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
</div>
```

The flex-basis Property

The **flex-basis** property specifies the initial length of a flex item.

Example

Set the initial length of the third flex item to 200 pixels:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-basis: 200px">3</div>
  <div>4</div>
</div>
```

The flex Property

The **flex** property is a shorthand property for the **flex-grow**, **flex-shrink**, and **flex-basis** properties.

Example

Make the third flex item not growable (0), not shrinkable (0), and with an initial length of 200 pixels:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex: 0 0 200px">3</div>
  <div>4</div>
</div>
```

The align-self Property

The **align-self** property specifies the alignment for the selected item inside the flexible container.

The **align-self** property overrides the default alignment set by the container's **align-items** property.

In these examples we use a 200 pixels high container, to better demonstrate the **align-self** property:

Example

Align the third flex item in the middle of the container:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="align-self: center">3</div>
  <div>4</div>
</div>
```

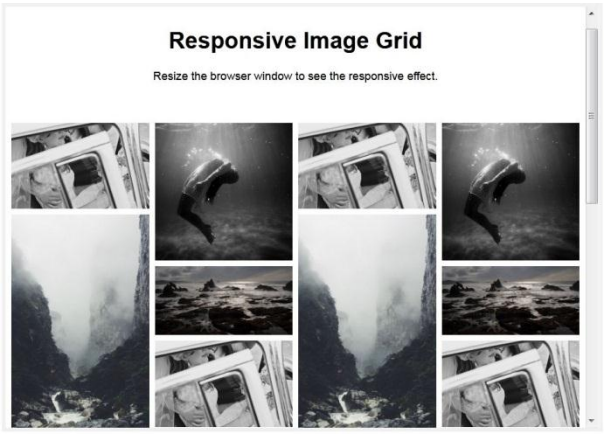
Example

Align the second flex item at the top of the container, and the third flex item at the bottom of the container:

```
<div class="flex-container">
  <div>1</div>
  <div style="align-self: flex-start">2</div>
  <div style="align-self: flex-end">3</div>
  <div>4</div>
</div>
```

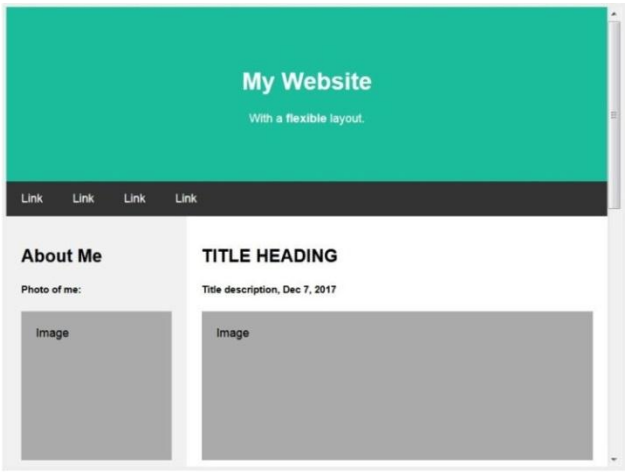
Responsive Image Gallery using Flexbox

Use flexbox to create a responsive image gallery that varies between four, two or full-width images, depending on screen size:



Responsive Website using Flexbox

Use flexbox to create a responsive website, containing a flexible navigation bar and flexible content:



CSS Flexbox Properties

The following table lists the CSS properties used with flexbox:

Property	Description
display	Specifies the type of box used for an HTML element
flex-direction	Specifies the direction of the flexible items inside a flex container
justify-content	Horizontally aligns the flex items when the items do not use all available space on the main-axis
align-items	Vertically aligns the flex items when the items do not use all available space on the cross-axis
flex-wrap	Specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line
align-content	Modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns
	flex lines
flex-flow	A shorthand property for flex-direction and flex-wrap
order	Specifies the order of a flexible item relative to the rest of the flex items inside the same container
align-self	Used on flex items. Overrides the container's align-items property
flex	A shorthand property for the flex-grow, flex-shrink, and the flex-basis properties

CSS Media Queries

CSS2 Introduced Media Types

The **@media** rule, introduced in CSS2, made it possible to define different style rules for different media types.

Examples: You could have one set of style rules for computer screens, one for printers, one for handheld devices, one for television-type devices, and so on.

Unfortunately these media types never got a lot of support by devices, other than the print media type.

CSS3 Introduced Media Queries

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

- width and height of the viewport
- width and height of the device
- orientation (is the tablet/phone in landscape or portrait mode?)
- resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

Browser Support

The numbers in the table specifies the first browser version that fully supports the **@media** rule.

Property					
@media	21.0	9.0	3.5	4.0	9.0

Media Query Syntax

A media query consists of a media type and can contain one or more expressions, which resolve to either true or false.

```
@media not|only mediatype and (expressions) {
  CSS-Code;
}
```

The result of the query is true if the specified media type matches the type of device the document is being displayed on and all expressions in the media query are true. When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules.

Unless you use the not or only operators, the media type is optional and the **all** type will be implied.

You can also have different stylesheets for different media:

```
<link rel="stylesheet" media="mediatype
and|not|only (expressions)" href="print.css">
```

CSS3 Media Types

Value	Description
all	Used for all media type devices
print	Used for printers
screen	Used for computer screens, tablets, smart-phones etc.
speech	Used for screenreaders that "reads" the page out loud

Media Queries Simple Examples

One way to use media queries is to have an alternate CSS section right inside your style sheet.

The following example changes the background-color to lightgreen if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the background-color will be pink):

Example

```
@media screen and (min-width: 480px) {
  body {
    background-color: lightgreen;
  }
}
```

The following example shows a menu that will float to the left of the page if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the menu will be on top of the content):

Example

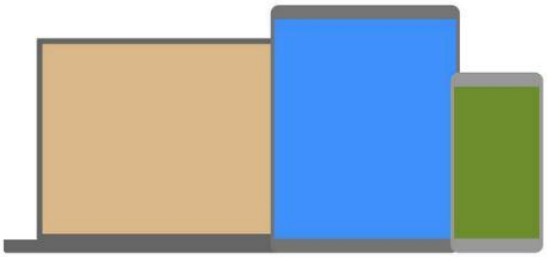
```
@media screen and (min-width: 480px) {
  #leftsidebar {width: 200px; float: left;}
  #main {margin-left: 216px;}
}
```

CSS Media Queries - Examples

CSS Media Queries - More Examples

Let us look at some more examples of using media queries.

Media queries are a popular technique for delivering a tailored style sheet to different devices. To demonstrate a simple example, we can change the background color for different devices:



Example

```
/* Set the background color of body to tan */
body {
  background-color: tan;
}

/* On screens that are 992px or less, set the
background color to blue */
@media screen and (max-width: 992px) {
  body {
    background-color: blue;
  }
}

/* On screens that are 600px or less, set the
background color to olive */
@media screen and (max-width: 600px) {
  body {
    background-color: olive;
  }
}
```

Do you wonder why we use exactly 992px and 600px? They are what we call "typical breakpoints" for devices. You can read more about typical breakpoints in our Responsive Web Design Tutorial.

Media Queries For Menus

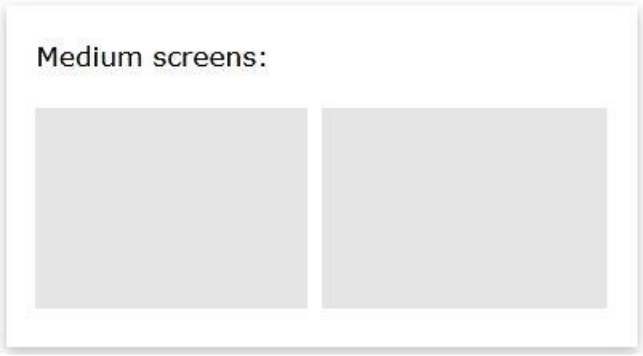
In this example, we use media queries to create a responsive navigation menu, that varies in design on different screen sizes.

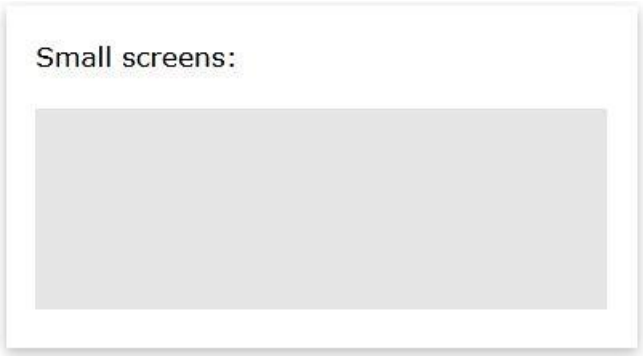
Example

```
/* The navbar container */
.topnav {
  overflow: hidden;
  background-color: #333;
}

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

/* On screens that are 600px wide or less, make
the menu links stack on top of each other instead
of next to each other */
@media screen and (max-width: 600px) {
  .topnav a {
    float: none;
    width: 100%;
  }
}
```

Media Queries For Columns

A common use of media queries, is to create a flexible layout. In this example, we create a layout that varies between four, two and full-width columns, depending on different screen sizes:





Example

```
/* Create four equal columns that floats next to
each other */
.column {
  float: left;
  width: 25%;
}

/* On screens that are 992px wide or less, go
from four columns to two columns */
@media screen and (max-width: 992px) {
  .column {
    width: 50%;
  }
}

/* On screens that are 600px wide or less, make
the columns stack on top of each other instead of
next to each other */
@media screen and (max-width: 600px) {
  .column {
    width: 100%;
  }
}
```

Tip: A more modern way of creating column layouts, is to use CSS Flexbox (see example below). However, it is not supported in Internet Explorer 10 and earlier versions. If you require IE6-10 support, use floats (as shown above).

Example

```
/* Container for flexboxes */
.row {
  display: flex;
  flex-wrap: wrap;
}

/* Create four equal columns */
.column {
  flex: 25%;
  padding: 20px;
}

/* On screens that are 992px wide or less, go
from four columns to two columns */
@media screen and (max-width: 992px) {
  .column {
    flex: 50%;
  }
}

/* On screens that are 600px wide or less, make
the columns stack on top of each other instead of
next to each other */
@media screen and (max-width: 600px) {
  .row {
    flex-direction: column;
  }
}
```

Hide Elements With Media Queries

Another common use of media queries, is to hide elements on different screen sizes:

Example

```
/* If the screen size is 600px wide or less, hide
the element */
@media screen and (max-width: 600px) {
  div.example {
    display: none;
  }
}
```

Change Font Size With Media Queries

You can also use media queries to change the font size of an element on different screen sizes:

Variable Font Size.

Example

```
/* If screen size is more than 600px wide, set
the font-size of <div> to 80px */
@media screen and (min-width: 600px) {
  div.example {
    font-size: 80px;
  }
}

/* If screen size is 600px wide, or less, set the
font-size of <div> to 30px */
@media screen and (max-width: 600px) {
  div.example {
    font-size: 30px;
  }
}
```

Flexible Image Gallery

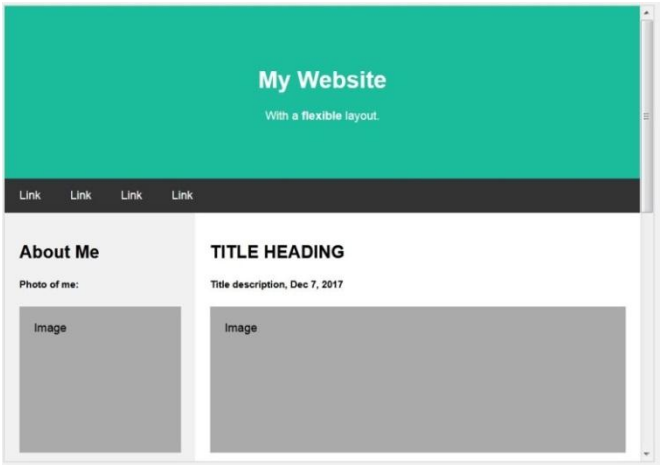
In this example, we use media queries together with flexbox to create a responsive image gallery:

Example

Flexible Website

In this example, we use media queries together with flexbox to create a responsive website, containing a flexible navigation bar and flexible content.

Example



Orientation: Portrait / Landscape

Media queries can also be used to change layout of a page depending on the orientation of the browser.

You can have a set of CSS properties that will only apply when the browser window is wider than its height, a so called "Landscape" orientation:

Example

Use a lightblue background color if the orientation is in landscape mode:

```
@media only screen and (orientation: landscape) {
  body {
    background-color: lightblue;
  }
}
```

Min Width to Max Width

You can also use the (**max-width: ..**) and (**min-width: ..**) values to set a minimum width and a maximum width.

For example, when the browser's width is between 600 and 900px, change the appearance of a element:

Example

```
@media screen and (max-width: 900px) and (min-
width: 600px) {
  div.example {
    font-size: 50px;
    padding: 50px;
    border: 8px solid black;
    background: yellow;
  }
}
```

Using an additional value: In the example below, we add an additional media query to our already existing one using a comma (this will behave like an OR operator):

Example

```
/* When the width is between 600px and 900px OR
above 1100px - change the appearance of <div> */
@media screen and (max-width: 900px) and (min-
width: 600px), (min-width: 1100px) {
  div.example {
    font-size: 50px;
    padding: 50px;
    border: 8px solid black;
    background: yellow;
  }
}
```

Responsive Web Design - Introduction

What is Responsive Web Design?

Responsive web design makes your web page look good on all devices.

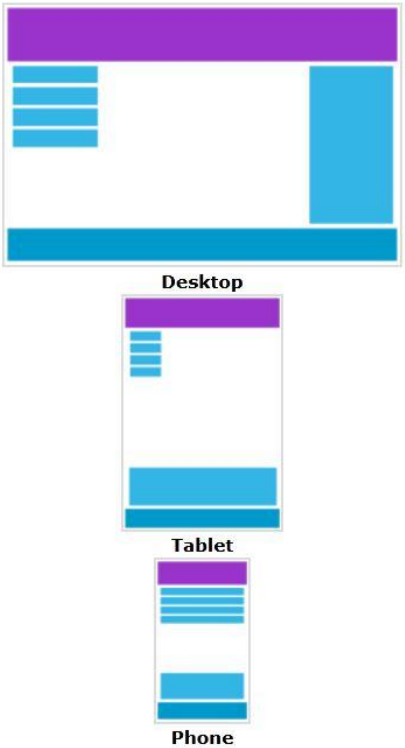
Responsive web design uses only HTML and CSS.

Responsive web design is not a program or a JavaScript.

Designing For The Best Experience For All Users

Web pages can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:



It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

Don't worry if you don't understand the example below, we will break down the code, step-by-step, in the next chapters:

Responsive Web Design - The Viewport

What is The Viewport?

The viewport is the user's visible area of a web page.

The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

This was not perfect!! But a quick fix.

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the **<meta>** tag.

You should include the following **<meta>** viewport element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

A **<meta>** viewport element gives the browser instructions on how to control the page's dimensions and scaling.

The **width=device-width** part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The **initial-scale=1.0** part sets the initial zoom level when the page is first loaded by the browser.

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport meta tag:

Tip: If you are browsing this page with a phone or a tablet, you can click on the two links below to see the difference.

Without the viewport meta tag

With the viewport meta tag

Size Content to The Viewport

Users are used to scroll websites vertically on both desktop and mobile devices - but not horizontally!

So, if the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience.

Some additional rules to follow:

1. Do NOT use large fixed width elements - For example, if an image is displayed at a width wider than the viewport it can cause the viewport to scroll horizontally. Remember to adjust this content to fit within the width of the viewport.

2. Do NOT let the content rely on a particular viewport width to render well - Since screen dimensions and width in CSS pixels vary widely between devices, content should not rely on a particular viewport width to render well.

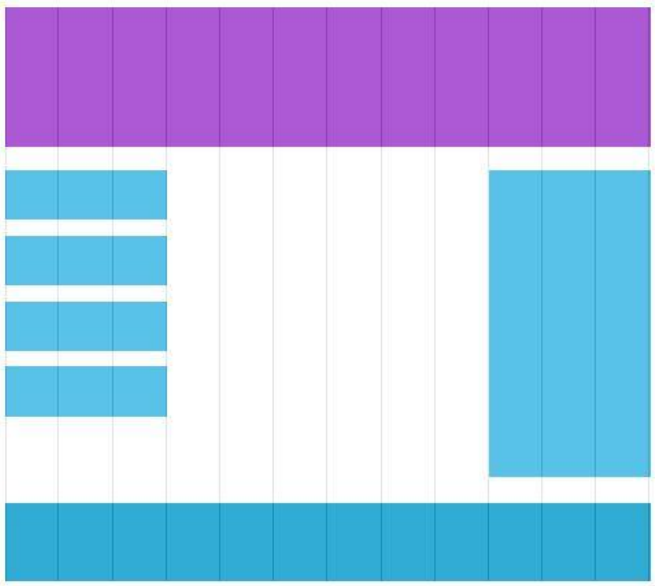
3. Use CSS media queries to apply different styling for small and large screens - Setting large absolute CSS widths for page elements will cause the element to be too wide for the viewport on a smaller device. Instead, consider using relative width values, such as width: 100%. Also, be careful of using large absolute positioning values. It may cause the element to fall outside the viewport on small devices.

Responsive Web Design - Grid-View

What is a Grid-View?

Many web pages are based on a grid-view, which means that the page is divided into columns:

Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.



A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

Building a Responsive Grid-View

Lets start building a responsive grid-view.

First ensure that all HTML elements have the **box-sizing** property set to **border-box**. This makes sure that the padding and border are included in the total width and height of the elements.

Add the following code in your CSS:

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

Read more about the **box-sizing** property in our CSS Box Sizing chapter.

The following example shows a simple responsive web page, with two columns:

25%	75%

Example

```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```

The example above is fine if the web page only contains two columns.

However, we want to use a responsive grid-view with 12 columns, to have more control over the web page.

First we must calculate the percentage for one column: 100% / 12 columns = 8.33%.

Then we make one class for each of the 12 columns, **class="col-"** and a number defining how many columns the section should span:

CSS:

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

All these columns should be floating to the left, and have a padding of 15px:

CSS:

```
[class*="col-"] {
  float: left;
  padding: 15px;
  border: 1px solid red;
}
```

Each row should be wrapped in a ** .** The number of columns inside a row should always add up to 12:

HTML:

```
<div class="row">
  <div class="col-3">...</div> <!-- 25% -->
  <div class="col-9">...</div> <!-- 75% -->
</div>
```

The columns inside a row are all floating to the left, and are therefore taken out of the flow of the page, and other elements will be placed as if the columns do not exist. To prevent this, we will add a style that clears the flow:

CSS:

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

We also want to add some styles and colors to make it look better:

Example

```
html {
  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color :#33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px
2px rgba(0,0,0,0.24);
}

.menu li:hover {
  background-color: #0099cc;
}
```

Notice that the webpage in the example does not look good when you resize the browser window to a very small width. In the next chapter you will learn how to fix that.

Responsive Web Design - Media Queries

What is a Media Query?

Media query is a CSS technique introduced in CSS3.

It uses the **@media** rule to include a block of CSS properties only if a certain condition is true.

Example

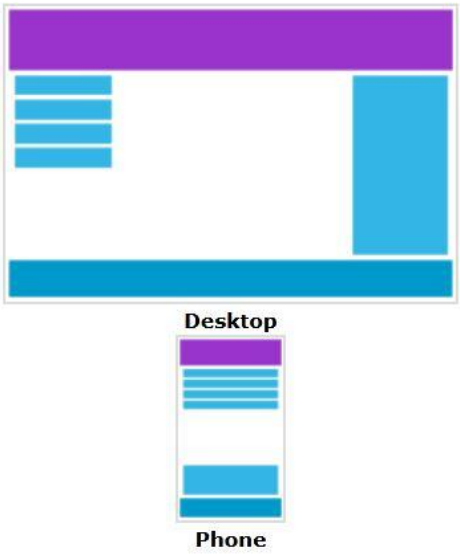
If the browser window is 600px or smaller, the background color will be lightblue:

```
@media only screen and (max-width: 600px) {
  body {
    background-color: lightblue;
  }
}
```

Add a Breakpoint

Earlier in this tutorial we made a web page with rows and columns, and it was responsive, but it did not look good on a small screen.

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.



Use a media query to add a breakpoint at 768px:

Example

When the screen (browser window) gets smaller than 768px, each column should have a width of 100%:

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
  /* For mobile phones: */
  [class*="col-"] {
    width: 100%;
  }
}
```

Always Design for Mobile First

Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices).

This means that we must make some changes in our CSS.

Instead of changing styles when the width gets *smaller* than 768px, we should change the design when the width gets *larger* than 768px. This will make our design Mobile First:

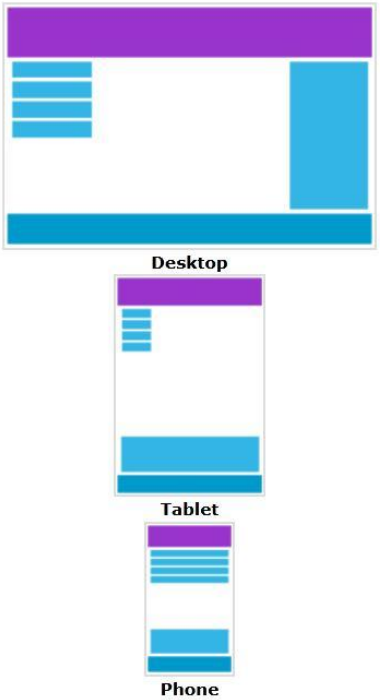
Example

```
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}

@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
```

Another Breakpoint

You can add as many breakpoints as you like.

We will also insert a breakpoint between tablets and mobile phones.



We do this by adding one more media query (at 600px), and a set of new classes for devices larger than 600px (but smaller than 768px):

Example

Note that the two sets of classes are almost identical, the only difference is the name (**col-** and **col-s-**):

```
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}

@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}

@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
```

It might seem odd that we have two sets of identical classes, but it gives us the opportunity *in HTML*, to decide what will happen with the columns at each breakpoint:

HTML Example

For desktop:

The first and the third section will both span 3 columns each. The middle section will span 6 columns.

For tablets:

The first section will span 3 columns, the second will span 9, and the third section will be displayed below the first two sections, and it will span 12 columns:

```
<div class="row">
  <div class="col-3 col-s-3">...</div>
  <div class="col-6 col-s-9">...</div>
  <div class="col-3 col-s-12">...</div>
</div>
```

Typical Device Breakpoints

There are tons of screens and devices with different heights and widths, so it is hard to create an exact breakpoint for each device. To keep things simple you could target five groups:

Example

```
/* Extra small devices (phones, 600px and down)
*/
@media only screen and (max-width: 600px) {...}

/* Small devices (portrait tablets and large
phones, 600px and up) */
@media only screen and (min-width: 600px) {...}

/* Medium devices (landscape tablets, 768px and
up) */
@media only screen and (min-width: 768px) {...}

/* Large devices (laptops/desktops, 992px and up)
*/
@media only screen and (min-width: 992px) {...}

/* Extra large devices (large laptops and
desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {...}
```

Orientation: Portrait / Landscape

Media queries can also be used to change layout of a page depending on the orientation of the browser.

You can have a set of CSS properties that will only apply when the browser window is wider than its height, a so called "Landscape" orientation:

Example

```
@media only screen and (orientation: landscape) {
  body {
    background-color: lightblue;
  }
}
```

Hide Elements With Media Queries

Another common use of media queries, is to hide elements on different screen sizes:

Example

```
/* If the screen size is 600px wide or less, hide
the element */
@media only screen and (max-width: 600px) {
  div.example {
    display: none;
  }
}
```

Change Font Size With Media Queries

You can also use media queries to change the font size of an element on different screen sizes:

Variable Font Size.

Example

```
/* If the screen size is 601px or more, set the
font-size of <div> to 80px */
@media only screen and (min-width: 601px) {
  div.example {
    font-size: 80px;
  }
}

/* If the screen size is 600px or less, set the
font-size of <div> to 30px */
@media only screen and (max-width: 600px) {
  div.example {
    font-size: 30px;
  }
}
```

CSS @media Reference

For a full overview of all the media types and features/expressions, please look at the @media rule in our CSS reference.

Responsive Web Design - Images

Resize the browser window to see how the image scales to fit the page.

Using The width Property

If the **width** property is set to a percentage and the height is set to "auto", the image will be responsive and scale up and down:

Example

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

Notice that in the example above, the image can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the **max-width** property instead.

Using The max-width Property

If the **max-width** property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

Example

```
img {
  max-width: 100%;
  height: auto;
}
```

Add an Image to The Example Web Page

Example

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

Background Images

Background images can also respond to resizing and scaling.

Here we will show three different methods:

1. If the **background-size** property is set to "contain", the background image will scale, and try to fit the content area. However, the image will keep its aspect ratio (the proportional relationship between the image's width and height):

Here is the CSS code:

Example

```
div {
  width: 100%;
  height: 400px;
  background-image: url('img_flowers.jpg');
  background-repeat: no-repeat;
  background-size: contain;
  border: 1px solid red;
}
```

2. If the **background-size** property is set to "100% 100%", the background image will stretch to cover the entire content area:

Here is the CSS code:

Example

```
div {
  width: 100%;
  height: 400px;
  background-image: url('img_flowers.jpg');
  background-size: 100% 100%;
  border: 1px solid red;
}
```

3. If the **background-size** property is set to "cover", the background image will scale to cover the entire content area. Notice that the "cover" value keeps the aspect ratio, and some part of the background image may be clipped:

Here is the CSS code:

Example

```
div {
  width: 100%;
  height: 400px;
  background-image: url('img_flowers.jpg');
  background-size: cover;
  border: 1px solid red;
}
```

Different Images for Different Devices

A large image can be perfect on a big computer screen, but useless on a small device. Why load a large image when you have to scale it down anyway? To reduce the load, or for any other reasons, you can use media queries to display different images on different devices.

Here is one large image and one smaller image that will be displayed on different devices:

Example

```
/* For width smaller than 400px: */
body {
  background-image: url('img_smallflower.jpg');
}

/* For width 400px and larger: */
@media only screen and (min-width: 400px) {
  body {
    background-image: url('img_flowers.jpg');
  }
}
```

You can use the media query **min-device-width**, instead of **min-width**, which checks the device width, instead of the browser width. Then the image will not change when you resize the browser window:

Example

```
/* For devices smaller than 400px: */
body {
  background-image: url('img_smallflower.jpg');
}

/* For devices 400px and larger: */
@media only screen and (min-device-width: 400px)
{
  body {
    background-image: url('img_flowers.jpg');
  }
}
```

HTML5 <picture> Element

HTML5 introduced the **<picture>** element, which lets you define more than one image.

Browser Support

Element					
<picture>	13	38.0	38.0	9.1	25.0

The **<picture>** element works similar to the **<video>** and **<audio>** elements. You set up different sources, and the first source that fits the preferences is the one being used:

Example

```
<picture>
  <source srcset="img_smallflower.jpg" media="
(max-width: 400px)">
  <source srcset="img_flowers.jpg">
  <img src="img_flowers.jpg" alt="Flowers">
</picture>
```

The **srcset** attribute is required, and defines the source of the image.

The **media** attribute is optional, and accepts the media queries you find in CSS @media rule.

You should also define an **<img>** element for browsers that do not support the **<picture>** element.

Responsive Web Design - Videos

Using The width Property

If the **width** property is set to 100%, the video player will be responsive and scale up and down:

Example

```
video {
  width: 100%;
  height: auto;
}
```

Notice that in the example above, the video player can be scaled up to be larger than its original size. A better solution, in many cases, will be to use the **max-width** property instead.

Using The max-width Property

If the **max-width** property is set to 100%, the video player will scale down if it has to, but never scale up to be larger than its original size:

Example

```
video {
  max-width: 100%;
  height: auto;
}
```

Add a Video to the Example Web Page

We want to add a video in our example web page. The video will be resized to always take up all the available space:

Example

```
video {
  width: 100%;
  height: auto;
}
```

Responsive Web Design - Frameworks

There are many existing CSS Frameworks that offer Responsive Design.

They are free, and easy to use.

Using W3.CSS

A great way to create a responsive design, is to use a responsive style sheet, like W3.CSS

W3.CSS makes it easy to develop sites that look nice at any size; desktop, laptop, tablet, or phone:

Example

```
<!DOCTYPE html>
<html>
<meta name="viewport" content="width=device-
width, initial-scale=1">
<link rel="stylesheet"
href="https://www.w3schools.com/w3css/4/w3.css">
<body>

<div class="w3-container w3-green">
  <h1>W3Schools Demo</h1>
  <p>Resize this responsive page!</p>
</div>

<div class="w3-row-padding">
  <div class="w3-third">
    <h2>London</h2>
    <p>London is the capital city of England.</p>
    <p>It is the most populous city in the United
Kingdom,
    with a metropolitan area of over 13 million
inhabitants.</p>
  </div>

  <div class="w3-third">
    <h2>Paris</h2>
    <p>Paris is the capital of France.</p>
    <p>The Paris area is one of the largest
population centers in Europe,
    with more than 12 million inhabitants.</p>
  </div>


  <div class="w3-third">
    <h2>Tokyo</h2>
    <p>Tokyo is the capital of Japan.</p>
    <p>It is the center of the Greater Tokyo
Area,
    and the most populous metropolitan area in
the world.</p>
  </div>
</div>

</body>
</html>
```

To learn more about W3.CSS, read our W3.CSS Tutorial.

Bootstrap

Another popular framework is Bootstrap, it uses HTML, CSS and jQuery to make responsive web pages.

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Bootstrap Example</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-
scale=1">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0
/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery
/3.3.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap
/3.4.0/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <div class="jumbotron">
    <h1>My First Bootstrap Page</h1>
  </div>
  <div class="row">
    <div class="col-sm-4">
      ...
    </div>
    <div class="col-sm-4">
      ...
    </div>
    <div class="col-sm-4">
    ...
    </div>
  </div>
</div>

</body>
</html>
```

To learn more about Bootstrap, go to our Bootstrap Tutorial.

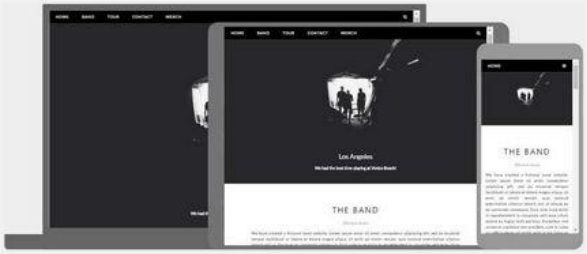
Responsive Web Design - Templates

W3.CSS Web Site Templates

We have created some responsive templates with the W3.CSS framework.

You are free to modify, save, share, and use them in all your projects.

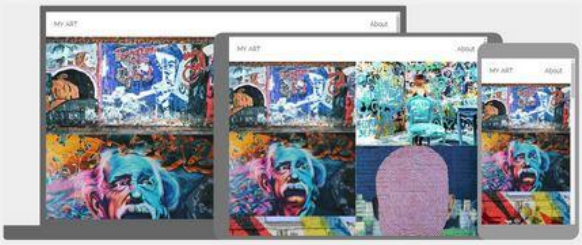
Band Template



Demo

Try it Yourself

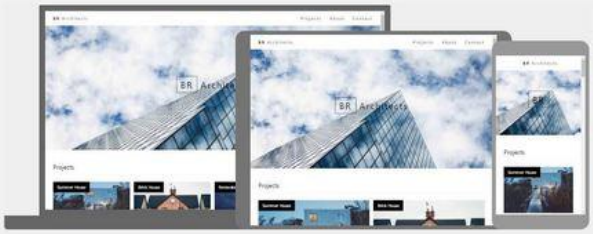
Art Template



Demo

Try it Yourself

Architect Template



Demo

Try it Yourself

Coming Soon Template

Demo

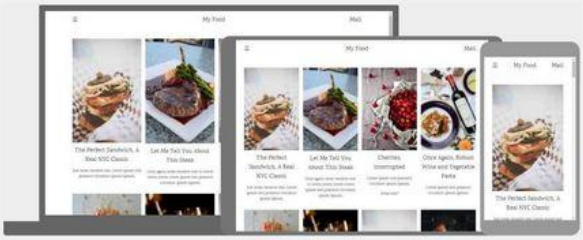
Try it Yourself

Blog Template

Demo

Try it Yourself

Food Blog Template



Demo

Try it Yourself

CV Template

Demo

Try it Yourself

Fashion Blog Template

Demo

Try it Yourself

Wedding Invitation Template

Demo

Try it Yourself

Gourmet Catering Template

Demo

Try it Yourself

Photo Template

Demo

Try it Yourself

Black & White Photo Template

Demo

Try it Yourself

People Portfolio Template

Demo

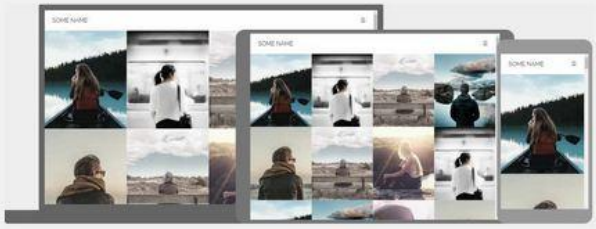
Try it Yourself

Photo III Template

Demo

Try it Yourself

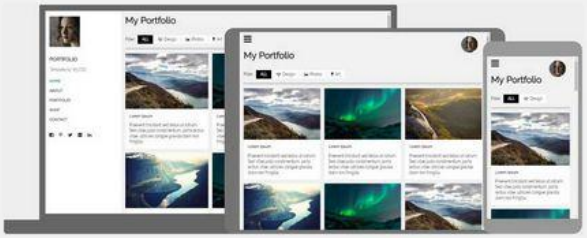
People Portfolio II Template



Demo

Try it Yourself

Nature Portfolio Template



Demo

Try it Yourself

Dark Portfolio Template

Demo

Try it Yourself

Black & White Portfolio Template

Demo

Try it Yourself

Interior Design Template

Demo

Try it Yourself

Parallax Template

Demo

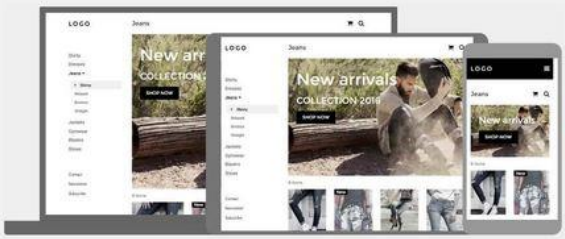
Try it Yourself

Cafe Template

Demo

Try it Yourself

Clothing Store Template



Demo

Try it Yourself

Pizza Restaurant Template

Demo

Try it Yourself

Modal Restaurant Template

Demo

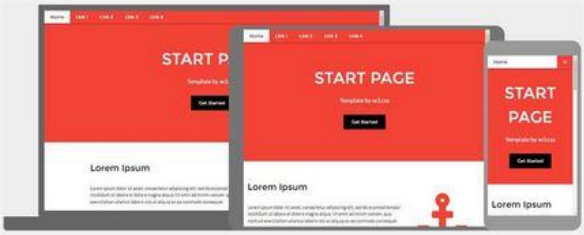
Try it Yourself

App Launch Template

Demo

Try it Yourself

Start Page Template



Demo

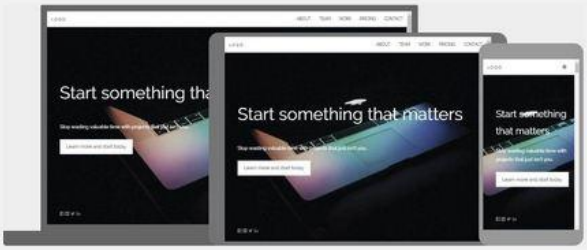
Try it Yourself

Marketing Template

Demo

Try it Yourself

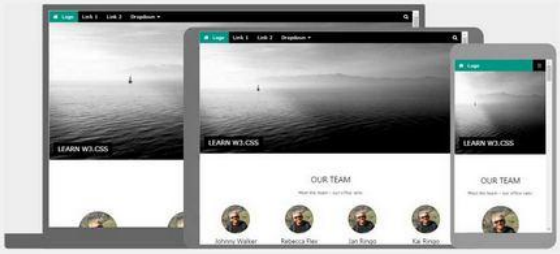
Startup Template



Demo

Try it Yourself

Marketing / Website Template



Demo

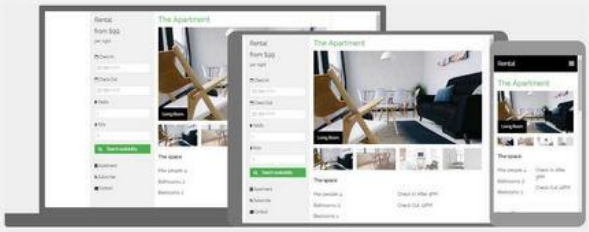
Try it Yourself

Web Page Template

Demo

Try it Yourself

Apartment Rental Template



Demo

Try it Yourself

Social Media Template

Demo

Try it Yourself

Hotel Template

Demo

Try it Yourself

Analytics Template

Demo

Try it Yourself

Travel Template

Demo

Try it Yourself

Travel Agency Template

Demo

Try it Yourself

Mail Template

Demo

Try it Yourself

House Design Template

Demo

Try it Yourself

Kitchen Sink/Demo Template

Black

Red

Teal

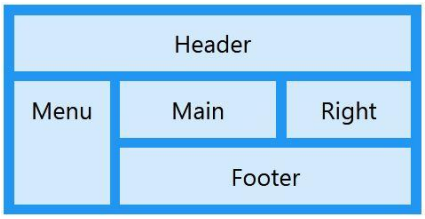
Try it Yourself

Screen 50/50 Template

Demo

Try it Yourself

CSS Grid Layout Module



Grid Layout

The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.

Browser Support

The grid properties are supported in all modern browsers.

57.0	16.0	52.0	10	44

Grid Elements

A grid layout consists of a parent element, with one or more child elements.

Example

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
  <div class="grid-item">7</div>
  <div class="grid-item">8</div>
  <div class="grid-item">9</div>
</div>
```

1	2	3
4	5	6
7	8	9

Display Property

An HTML element becomes a grid container when its **display** property is set to *grid* or *inline-grid*.

Example

```
.grid-container {
  display: grid;
}
```

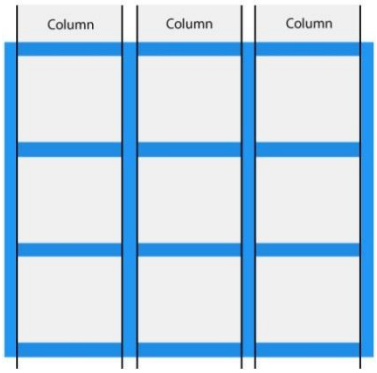
Example

```
.grid-container {
  display: inline-grid;
}
```

All direct children of the grid container automatically become *grid items*.

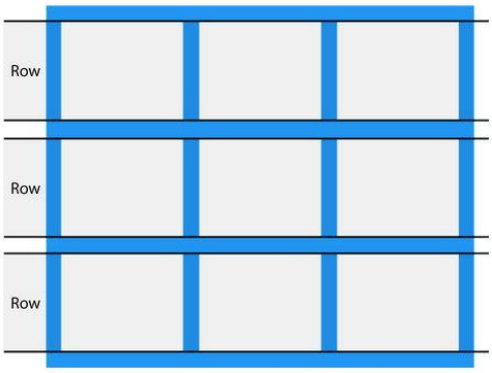
Grid Columns

The vertical lines of grid items are called *columns*.



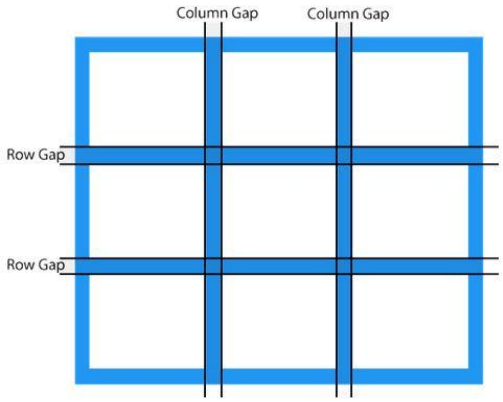
Grid Rows

The horizontal lines of grid items are called *rows*.



Grid Gaps

The spaces between each column/row are called *gaps*.



You can adjust the gap size by using one of the following properties:

```
grid-column-gap
grid-row-gap
grid-gap
```

Example

The *grid-column-gap* property sets the gap between the columns:

```
.grid-container {
  display: grid;
  grid-column-gap: 50px;
}
```

Example

The *grid-row-gap* property sets the gap between the rows:

```
.grid-container {
  display: grid;
  grid-row-gap: 50px;
}
```

Example

The *grid-gap* property is a shorthand property for the *grid-column-gap* and the *grid-row-gap* properties:

```
.grid-container {
  display: grid;
  grid-gap: 50px 100px;
}
```

Example

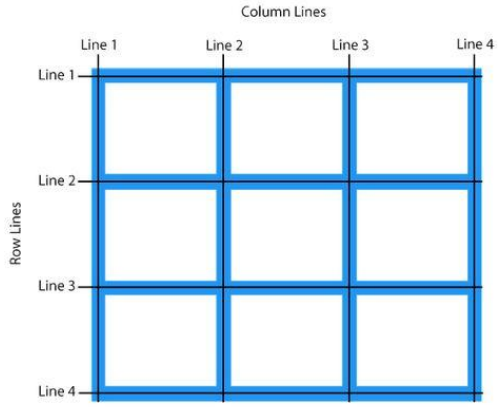
The *grid-gap* property can also be used to set both the row gap and the column gap in one value:

```
.grid-container {
  display: grid;
  grid-gap: 50px;
}
```

Grid Lines

The lines between columns are called *column lines*.

The lines between rows are called *row lines*.



Refer to line numbers when placing a grid item in a grid container:

Example

Place a grid item at column line 1, and let it end on column line 3:

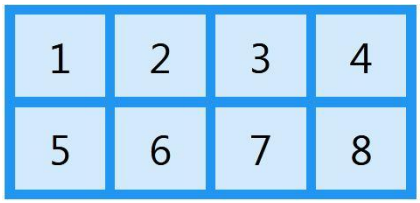
```
.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

Example

Place a grid item at row line 1, and let it end on row line 3:

```
.item1 {
  grid-row-start: 1;
  grid-row-end: 3;
}
```

CSS Grid Container



Grid Container

To make an HTML element behave as a grid container, you have to set the display property to *grid* or *inline-grid*.

Grid containers consist of grid items, placed inside columns and rows.

The grid-template-columns Property

The **grid-template-columns** property defines the number of columns in your grid layout, and it can define the width of each column.

The value is a space-separated-list, where each value defines the length of the respective column.

If you want your grid layout to contain 4 columns, specify the width of the 4 columns, or "auto" if all columns should have the same width.

Example

Make a grid with 4 columns:

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
}
```

Note: If you have more than 4 items in a 4 columns grid, the grid will automatically add a new row to put the items in.

The **grid-template-columns** property can also be used to specify the size (width) of the columns.

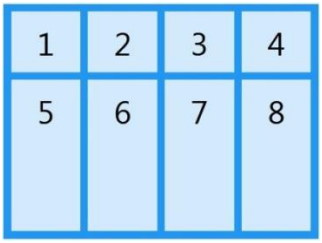
Example

Set a size for the 4 columns:

```
.grid-container {
  display: grid;
  grid-template-columns: 80px 200px auto 40px;
}
```

The grid-template-rows Property

The **grid-template-rows** property defines the height of each row.



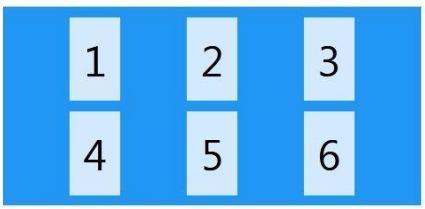
The value is a space-separated-list, where each value defines the height of the respective row:

Example

```
.grid-container {
  display: grid;
  grid-template-rows: 80px 200px;
}
```

The justify-content Property

The **justify-content** property is used to align the whole grid inside the container.



Note: The grid's total width has to be less than the container's width for the justify-content property to have any effect.

Example

```
.grid-container {
  display: grid;
  justify-content: space-evenly;
}
```

Example

```
.grid-container {
  display: grid;
  justify-content: space-around;
}
```

Example

```
.grid-container {
  display: grid;
  justify-content: space-between;
}
```

Example

```
.grid-container {
  display: grid;
  justify-content: center;
}
```

Example

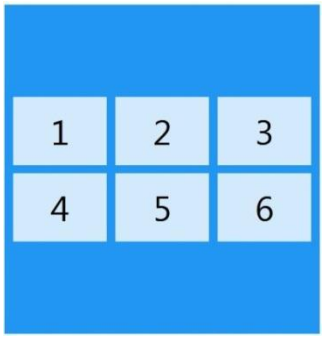
```
.grid-container {
  display: grid;
  justify-content: start;
}
```

Example

```
.grid-container {
  display: grid;
  justify-content: end;
}
```

The align-content Property

The **align-content** property is used to *vertically* align the whole grid inside the container.



Note: The grid's total height has to be less than the container's height for the align-content property to have any effect.

Example

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: center;
}
```

Example

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-evenly;
}
```

Example

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-around;
}
```

Example

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: space-between;
}
```

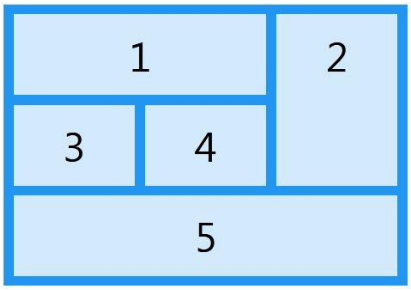
Example

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: start;
}
```

Example

```
.grid-container {
  display: grid;
  height: 400px;
  align-content: end;
}
```

CSS Grid Item



Child Elements (Items)

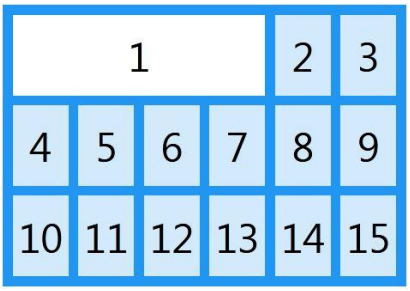
A grid *container* contains grid *items*.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

The grid-column Property:

The **grid-column** property defines on which column(s) to place an item.

You define where the item will start, and where the item will end.



Note: The *grid-column* property is a shorthand property for the *grid-column-start* and the *grid-column-end* properties.

To place an item, you can refer to *line numbers*, or use the keyword "span" to define how many columns the item will span.

Example

Make "item1" start on column 1 and end before column 5:

```
.item1 {
  grid-column: 1 / 5;
}
```

Example

Make "item1" start on column 1 and span 3 columns:

```
.item1 {
  grid-column: 1 / span 3;
}
```

Example

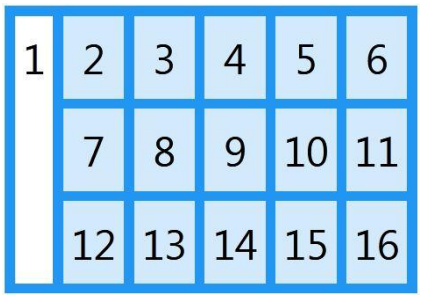
Make "item2" start on column 2 and span 3 columns:

```
.item2 {
  grid-column: 2 / span 3;
}
```

The grid-row Property:

The **grid-row** property defines on which row to place an item.

You define where the item will start, and where the item will end.



Note: The *grid-row* property is a shorthand property for the *grid-row-start* and the *grid-row-end* properties.

To place an item, you can refer to *line numbers*, or use the keyword "span" to define how many rows the item will span:

Example

Make "item1" start on row-line 1 and end on row-line 4:

```
.item1 {
  grid-row: 1 / 4;
}
```

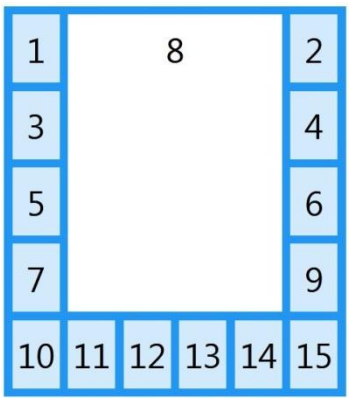
Example

Make "item1" start on row 1 and span 2 rows:

```
.item1 {
  grid-row: 1 / span 2;
}
```

The grid-area Property

The **grid-area** property can be used as a shorthand property for the *grid-row-start*, *grid-column-start*, *grid-row-end* and the *grid-column-end* properties.



Example

Make "item8" start on row-line 1 and column-line 2, and end on row-line 5 and column line 6:

```
.item8 {
  grid-area: 1 / 2 / 5 / 6;
}
```

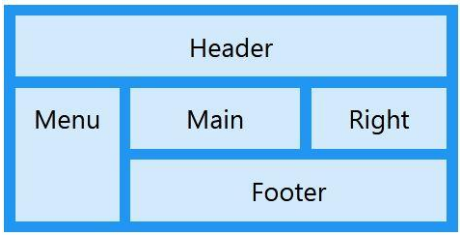
Example

Make "item8" start on row-line 2 and column-line 1, and span 2 rows and 3 columns:

```
.item8 {
  grid-area: 2 / 1 / span 2 / span 3;
}
```

Naming Grid Items

The **grid-area** property can also be used to assign names to grid items.



Named grid items can be referred to by the **grid-template-areas** property of the grid container.

Example

Item1 gets the name "myArea" and spans all five columns in a five columns grid layout:

```
.item1 {
  grid-area: myArea;
}
.grid-container {
  grid-template-areas: 'myArea myArea myArea
myArea myArea';
}
```

Each row is defined by apostrophes (' ')

The columns in each row is defined inside the apostrophes, separated by a space.

Note: A period sign represents a grid item with no name.

Example

Let "myArea" span two columns in a five columns grid layout (period signs represent items with no name):

```
.item1 {
  grid-area: myArea;
}
.grid-container {
  grid-template-areas: 'myArea myArea . . .';
}
```

To define two rows, define the column of the second row inside another set of apostrophes:

Example

Make "item1" span two columns *and* two rows:

```
.grid-container {
  grid-template-areas: 'myArea myArea . . .'
'myArea myArea . . .';
}
```

Example

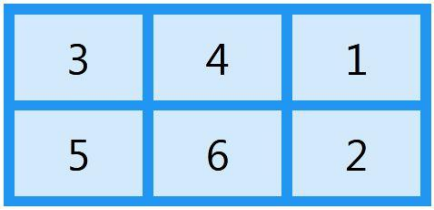
Name all items, and make a ready-to-use webpage template:

```
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }

.grid-container {
  grid-template-areas:
    'header header header header header header'
    'menu main main main right right'
    'menu footer footer footer footer footer';
}
```

The Order of the Items

The Grid Layout allows us to position the items anywhere we like.

The first item in the HTML code does not have to appear as the first item in the grid.



Example

```
.item1 { grid-area: 1 / 3 / 2 / 4; }
.item2 { grid-area: 2 / 3 / 3 / 4; }
.item3 { grid-area: 1 / 1 / 2 / 2; }
.item4 { grid-area: 1 / 2 / 2 / 3; }
.item5 { grid-area: 2 / 1 / 3 / 2; }
.item6 { grid-area: 2 / 2 / 3 / 3; }
```

You can re-arrange the order for certain screen sizes, by using media queries:

Example

```
@media only screen and (max-width: 500px) {
  .item1 { grid-area: 1 / span 3 / 2 / 4; }
  .item2 { grid-area: 3 / 3 / 4 / 4; }
  .item3 { grid-area: 2 / 1 / 3 / 2; }
  .item4 { grid-area: 2 / 2 / span 2 / 3; }
  .item5 { grid-area: 3 / 1 / 4 / 2; }
  .item6 { grid-area: 2 / 3 / 3 / 4; }
}
```