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!

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



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

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

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

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

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](https://www.w3schools.com/css/css_combinators.asp) (select elements based on a specific relationship between them)
* [Pseudo-class selectors](https://www.w3schools.com/css/css_pseudo_classes.asp) (select elements based on a certain state)
* [Pseudo-elements selectors](https://www.w3schools.com/css/css_pseudo_elements.asp) (select and style a part of an element)
* [Attribute selectors](https://www.w3schools.com/css/css_attribute_selectors.asp) (select elements based on an attribute or attribute value)

## 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.

#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.

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

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

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

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

HTML elements can also refer to more than one class.

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

<style>

p.center {

text-align: center;

color: red;

}

p.large {

font-size: 300%;

}

</style>

</head>

<body>

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

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

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

**Note:** A class name cannot start with a number!

## The CSS Universal Selector

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

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

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

To group selectors, separate each selector with a comma. In this example we have grouped the selectors from the code above:

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

Three Ways to Insert CSS

* 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.

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

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

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

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

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

### **"mystyle.css"**

body {  
  background-color: lightblue;  
}  
  
h1 {  
  color: navy;  
  margin-left: 20px;  
}

**Note:** Do not add a space between the property value (20) and the unit (px):  
Incorrect (space): margin-left: 20 px;  
Correct (no space): margin-left: 20px;

## Internal CSS

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

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

### **Example**

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

<!DOCTYPE html>  
<html>  
<head>  
<style>  
body {  
  background-color: linen;  
}  
  
h1 {  
  color: maroon;  
  margin-left: 40px;  
}  
</style>  
</head>  
<body>

## 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.

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

**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;  
}

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>

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

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

### **Example**

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

## Opacity / Transparency

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

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

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

## Transparency using RGBA

If you do not want to apply opacity to child elements, like in our example above, use **RGBA** color values.

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).

div {  
  background: rgba(0, 128, 0, 0.3) /\* Green background with 30% opacity \*/  
}

**Note:** When using a background image, use an image that does not disturb the text. The background image can also be set for specific elements, like the <p> element:

### **Example**

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

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;

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

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

## CSS background-position

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

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

## CSS background-attachment

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

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;  
}

CSS background - Shorthand property

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

Instead of writing:

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

You can use the shorthand property background: Use the shorthand property to set 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

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

## 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):

<style>

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 \*/

}

</style>

## 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).

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

## HEX Values

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

## RGB Values

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

## HSL Values

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

## 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):

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

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

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

## CSS Rounded Borders

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

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

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.

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

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

If the margin property has two values:

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

If the margin property has one value:

* **margin: 25px;**

all four margins are 25px

## The auto Value

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

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

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 (<div>):

div {  
  border: 1px solid red;  
  margin-left: 100px;  
}  
  
p.ex1 {  
  margin-left: inherit;  
}

<div>

<p class="ex1">This paragraph has an inherited left margin (from the div element).</p>

</div>

## 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:

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.

## 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.

Set different padding for all four sides of a <div> 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

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

If the padding property has two values:

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

If the padding property has one value:

* **padding: 25px;**
  + all four paddings are 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](https://www.w3schools.com/css/css_boxmodel.asp)).

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.

Here, the <div> element is given a width of 300px. However, the actual width of the <div> 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 actual width; if you increase the padding, the available content space will decrease. 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;  
}

## 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 and width Values

The height and width properties may have the following values:

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

## Setting max-width

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

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

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

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

**Note:** If you for some reason use both the width property and the max-width property on the same element, and the value of the width property is larger than the max-width property; the max-width property will be used (and the width property will be ignored).

The CSS Box Model

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

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

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

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

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

Here is the calculation:

  320px (width of content area)  
+ 20px (left padding + right padding)  
+ 10px (left border + right border)  
**= 350px (total width)**  
  
  50px (height of content area)  
+ 20px (top padding + bottom padding)  
+ 10px (top border + bottom border)  
**= 80px (total height)**

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

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

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

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

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

CSS has the following outline properties:

* outline-style
* outline-color
* outline-width
* outline-offset
* outline

**Note:** Outline differs from [borders](https://www.w3schools.com/css/css_border.asp)! 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

**Note:** None of the other outline properties (which you will learn more about in the next chapters) will have ANY effect unless the outline-style property is set!

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

CSS Outline - Shorthand property

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

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

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

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

## CSS Outline Offset

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

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

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

## 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](https://www.w3schools.com/cssref/css_colors_legal.asp) for a complete list of possible color values.

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

## Text Color and Background Color

 we define both the background-color property and the color property

**Important:** High contrast is very important for people with vision problems. So, always ensure that the contrast between the text color and the background color (or background image) is good!

Text Alignment and Text Direction

In this chapter you will learn about the following properties:

* text-align
* text-align-last
* direction
* unicode-bidi
* vertical-align

## Text Alignment

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

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

When the text-align property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight

div {  
  text-align: justify;  
}

## Text Align Last

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

Align the last line of text in three <p> elements:

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

## Text Direction

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

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

## Vertical Alignment

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

<style>

img.a {

vertical-align: baseline;

}

img.b {

vertical-align: text-top;

}

img.c {

vertical-align: text-bottom;

}

img.d {

vertical-align: sub;

}

img.e {

vertical-align: super;

}

</style>

</head>

<body>

<h1>The vertical-align Property</h1>

<h2>vertical-align: baseline (default):</h2>

<p>An <img class="a" src="sqpurple.gif" width="9" height="9"> image with a default alignment.</p>

<h2>vertical-align: text-top:</h2>

<p>An <img class="b" src="sqpurple.gif" width="9" height="9"> image with a text-top alignment.</p>

<h2>vertical-align: text-bottom:</h2>

<p>An <img class="c" src="sqpurple.gif" width="9" height="9"> image with a text-bottom alignment.</p>

<h2>vertical-align: sub:</h2>

<p>An <img class="d" src="sqpurple.gif" width="9" height="9"> image with a sub alignment.</p>

<h2>vertical-align: sup:</h2>

<p>An <img class="e" src="sqpurple.gif" width="9" height="9"> image with a super alignment.</p>

</body>

</html>

Text Decoration

In this chapter you will learn about the following properties:

* text-decoration-line
* text-decoration-color
* text-decoration-style
* text-decoration-thickness
* text-decoration

## Add a Decoration Line to Text

The text-decoration-line property is used to add a decoration line to text.

**Tip:** You can combine more than one value, like overline and underline to display lines both over and under a text.

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

## Specify a Style for the Decoration Line

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

h1 {  
  text-decoration-line: underline;  
  text-decoration-style: solid;  
}  
  
h2 {  
  text-decoration-line: underline;  
  text-decoration-style: double;  
}

## Specify the Thickness for the Decoration Line

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

The Shorthand Property

The text-decoration property is a shorthand property for:

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

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

All links in HTML are underlined by default. Sometimes you see that links are styled with no underline. The text-decoration: none; is used to remove the underline from links, like this:

a {  
  text-decoration: none;  
}

## Text Transformation

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

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

p.uppercase {  
  text-transform: uppercase;  
}  
  
p.lowercase {  
  text-transform: lowercase;  
}  
  
p.capitalize {  
  text-transform: capitalize;  
}

Text Spacing

In this chapter you will learn about the following properties:

* text-indent
* letter-spacing
* line-height
* word-spacing
* white-space

## Text Indentation

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

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:

h1 {  
  letter-spacing: 5px;  
}  
  
h2 {  
  letter-spacing: -2px;  
}

## Line Height

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

p.small {  
  line-height: 0.8;  
}  
  
p.big {  
  line-height: 1.8;  
}

## 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:

p.one {  
  word-spacing: 10px;  
}  
  
p.two {  
  word-spacing: -2px;  
}

## White Space

The white-space property specifies how white-space inside an element is handled.

This example demonstrates how to disable text wrapping inside an element:

p {  
  white-space: nowrap;  
}

## Text Shadow

The text-shadow property adds shadow to text.

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

h1 {  
  text-shadow: 2px 2px;  
}

**Tip:** [Go to our CSS Fonts](https://www.w3schools.com/css/css_font.asp) chapter to learn about how to change fonts, text size and the style of a text.

**Tip:** [Go to our CSS Text Effects](https://www.w3schools.com/css/css3_text_effects.asp) chapter to learn about different text effects.

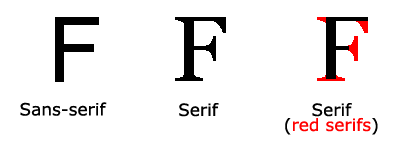
Generic Font Families

In CSS there are five generic font families:

1. **Serif** fonts have a small stroke at the edges of each letter. They create a sense of formality and elegance.
2. **Sans-serif** fonts have clean lines (no small strokes attached). They create a modern and minimalistic look.
3. **Monospace** fonts - here all the letters have the same fixed width. They create a mechanical look.
4. **Cursive** fonts imitate human handwriting.
5. **Fantasy** fonts are decorative/playful fonts.

All the different font names belong to one of the generic font families.

Difference Between Serif and Sans-serif Fonts



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

Some Font Examples

|  |  |
| --- | --- |
| **Generic Font Family** | **Examples of Font Names** |
| Serif | Times New Roman Georgia Garamond |
| Sans-serif | Arial Verdana Helvetica |
| Monospace | Courier New Lucida Console Monaco |
| Cursive | Brush Script MT Lucida Handwriting |
| Fantasy | Copperplate Papyrus |

## The CSS font-family Property

In CSS, we use the font-family property to specify the font of a text.

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

Specify some different fonts for three paragraphs:

.p1 {  
  font-family: "Times New Roman", Times, serif;  
}  
  
.p2 {  
  font-family: Arial, Helvetica, sans-serif;  
}  
  
.p3 {  
  font-family: "Lucida Console", "Courier New", monospace;  
}

## What are Web Safe Fonts?

Web safe fonts are fonts that are universally installed across all browsers and devices.

## Fallback Fonts

However, there are no 100% completely web safe fonts. There is always a chance that a font is not found or is not installed properly.

Therefore, it is very important to always use fallback fonts.

This means that you should add a list of similar "backup fonts" in the font-family property. If the first font does not work, the browser will try the next one, and the next one, and so on. Always end the list with a generic font family name.

Here, there are three font types: Tahoma, Verdana, and sans-serif. The second and third fonts are backups, in case the first one is not found.

p {  
font-family: Tahoma, Verdana, sans-serif;  
}

Best Web Safe Fonts for HTML and CSS

The following list are the best web safe fonts for HTML and CSS:

* Arial (sans-serif)
* Verdana (sans-serif)
* Tahoma (sans-serif)
* Trebuchet MS (sans-serif)
* Times New Roman (serif)
* Georgia (serif)
* Garamond (serif)
* Courier New (monospace)
* Brush Script MT (cursive)

**Note:** Before you publish your website, always check how your fonts appear on different browsers and devices, and always use [fallback fonts](https://www.w3schools.com/css/css_font_fallbacks.asp)!

## Arial (sans-serif)

Arial is the most widely used font for both online and printed media. Arial is also the default font in Google Docs.

Arial is one of the safest web fonts, and it is available on all major operating systems.

## Verdana (sans-serif)

Verdana is a very popular font. Verdana is easily readable even for small font sizes.

Commonly Used Font Fallbacks

Below are some commonly used font fallbacks, organized by the 5 generic font families:

* **Serif**
* **Sans-serif**
* **Monospace**
* **Cursive**
* **Fantasy**

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)

## Font Weight

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

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

## 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.

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

## Set Font Size With Pixels

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

h1 {  
  font-size: 40px;  
}

**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.

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*

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

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

## Google Fonts

If you do not want to use any of the standard fonts in HTML, you can use Google Fonts.

Google Fonts are free to use, and have more than 1000 fonts to choose from.

## How To Use Google Fonts

Just add a special style sheet link in the <head> section and then refer to the font in the CSS.

Here, we want to use a font named "Sofia" from Google Fonts:

<head>  
**<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">**  
<style>  
body {  
  font-family: "Sofia", sans-serif;  
}  
</style>  
</head>

Here, we want to use a font named "Audiowide" from Google Fonts:

<head>  
**<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Audiowide">**  
<style>  
body {  
  font-family: "Audiowide", sans-serif;  
}  
</style>  
</head>

## Use Multiple Google Fonts

To use multiple Google fonts, just separate the font names with a pipe character (|), like this:

Request multiple fonts:

<head>  
**<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Audiowide|Sofia|Trirong">**  
<style>  
h1.a {font-family: "Audiowide", sans-serif;}  
h1.b {font-family: "Sofia", sans-serif;}  
h1.c {font-family: "Trirong", serif;}  
</style>  
</head>

**Note:** Requesting multiple fonts may slow down your web pages! So be careful about that.

To request multiple font effects, just separate the effect names with a pipe character (|), like this: Add multiple effects to the "Sofia" font:

<head>  
**<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia&effect=neon|outline|emboss|shadow-multiple">**  
<style>  
body {  
  font-family: "Sofia", sans-serif;  
  font-size: 30px;  
}  
</style>  
</head>  
<body>  
  
<h1 class="font-effect-neon">Neon Effect</h1>  
<h1 class="font-effect-outline">Outline Effect</h1>  
<h1 class="font-effect-emboss">Emboss Effect</h1>  
<h1 class="font-effect-shadow-multiple">Multiple Shadow Effect</h1>  
  
</body>

## Font Pairing Rules

Here are some basic rules to create great font pairings:

### **1. Complement**

It is always safe to find font pairings that complement one another.

A great font combination should harmonize, without being too similar or too different.

### **2. Use Font Superfamilies**

A font superfamily is a set of fonts designed to work well together. So, using different fonts within the same superfamily is safe.

For example, the Lucida superfamily contains the following fonts: Lucida Sans, Lucida Serif, Lucida Typewriter Sans, Lucida Typewriter Serif and Lucida Math.

### **3. Contrast is King**

Two fonts that are too similar will often conflict. However, contrasts, done the right way, brings out the best in each font.

Example: Combining serif with sans serif is a well known combination.

A strong superfamily includes both serif and sans serif variations of the same font (e.g. Lucida and Lucida Sans).

### **4. Choose Only One Boss**

One font should be the boss. This establishes a hierarchy for the fonts on your page. This can be achieved by varying the size, weight and color.

No doubt "Georgia" is the boss here:

body {  
  background-color: black;  
  font-family: Verdana, sans-serif;  
  font-size: 16px;  
  color: gray;  
}  
  
h1 {  
  font-family: Georgia, serif;  
  font-size: 60px;  
  color: white;  
}

The CSS Font Property

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

The font property is a shorthand property for:

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

**Note:** The font-size and font-family values are required. If one of the other values is missing, their default value are used.

## 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 <span>).

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](https://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" crossorigin="anonymous"></script>

Read more about how to get started with Font Awesome in our [Font Awesome 5 tutorial](https://www.w3schools.com/icons/fontawesome5_intro.asp).

**Note:** No downloading or installation is required!

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

## 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!

## 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">

# **CSS Links**

With CSS, links can be styled in many different ways.

## Styling Links

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

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

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

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

## Text Decoration

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

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

## Background Color

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

a:link {  
  background-color: yellow;  
}

## Styling Links

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

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
* This example demonstrates how to add other styles to hyperlinks:
* a.one:link {color: #ff0000;}  
  a.one:visited {color: #0000ff;}  
  a.one:hover {color: #ffcc00;}  
    
  a.two:link {color: #ff0000;}  
  a.two:visited {color: #0000ff;}  
  a.two:hover {font-size: 150%;}  
    
  a.three:link {color: #ff0000;}  
  a.three:visited {color: #0000ff;}  
  a.three:hover {background: #66ff66;}  
    
  a.four:link {color: #ff0000;}  
  a.four:visited {color: #0000ff;}  
  a.four:hover {font-family: monospace;}  
    
  a.five:link {color: #ff0000; text-decoration: none;}  
  a.five:visited {color: #0000ff; text-decoration: none;}  
  a.five:hover {text-decoration: underline;}
* This example demonstrates the different types of cursors (can be useful for links):
* <span style="cursor: auto">auto</span><br>  
  <span style="cursor: crosshair">crosshair</span><br>  
  <span style="cursor: default">default</span><br>  
  <span style="cursor: e-resize">e-resize</span><br>  
  <span style="cursor: help">help</span><br>  
  <span style="cursor: move">move</span><br>  
  <span style="cursor: n-resize">n-resize</span><br>  
  <span style="cursor: ne-resize">ne-resize</span><br>  
  <span style="cursor: nw-resize">nw-resize</span><br>  
  <span style="cursor: pointer">pointer</span><br>  
  <span style="cursor: progress">progress</span><br>  
  <span style="cursor: s-resize">s-resize</span><br>  
  <span style="cursor: se-resize">se-resize</span><br>  
  <span style="cursor: sw-resize">sw-resize</span><br>  
  <span style="cursor: text">text</span><br>  
  <span style="cursor: w-resize">w-resize</span><br>  
  <span style="cursor: wait">wait</span>

## 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.

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

## An Image as The List Item Marker

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

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

Position The List Item Markers

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

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

* 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

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>:

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:

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 is 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:

ol {  
  background: #ff9999;  
  padding: 20px;  
}  
  
ul {  
  background: #3399ff;  
  padding: 20px;  
}  
  
ol li {  
  background: #ffe5e5;  
  color: darkred;  
  padding: 5px;  
  margin-left: 35px;  
}  
  
ul li {  
  background: #cce5ff;  
  color: darkblue;  
  margin: 5px;  
}

## Full-Width Table

The table above might seem small in some cases. If you need a table that should span the entire screen (full-width), add width: 100% to the <table> element

### **Double Borders**

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

To remove double borders, take a look at the example below.

## Collapse Table Borders

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

table {  
  border-collapse: collapse;  
}

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

table {  
  border: 1px solid;  
}

To create a table that should only span half the page, use width: 50%:

table {  
  width: 50%;  
}

## 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.

To center-align the content of  <td> elements as well, use text-align: center

## 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).

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:

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

## Horizontal Dividers

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

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

## Hoverable Table

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

tr:hover {background-color: coral;}

## Striped Tables

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

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

## Responsive Table

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

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

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

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

## The display Property

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

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

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

Block-level Elements

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

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

Examples of block-level elements:

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

Inline Elements

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

This is an inline <span> element inside a paragraph.

Examples of inline elements:

* <span>
* <a>
* <img>

|  |  |
| --- | --- |
| **Value** | **Description** |
| inline | Displays an element as an inline element |
| block | Displays an element as a block element |
| contents | Makes the container disappear, making the child elements children of the element the next level up in the DOM |
| flex | Displays an element as a block-level flex container |
| grid | Displays an element as a block-level grid container |
| inline-block | Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values |
| inline-flex | Displays an element as an inline-level flex container |
| inline-grid | Displays an element as an inline-level grid container |
| inline-table | The element is displayed as an inline-level table |
| list-item | Let the element behave like a <li> element |
| run-in | Displays an element as either block or inline, depending on context |
| table | Let the element behave like a <table> element |
| table-caption | Let the element behave like a <caption> element |
| table-column-group | Let the element behave like a <colgroup> element |
| table-header-group | Let the element behave like a <thead> element |
| table-footer-group | Let the element behave like a <tfoot> element |
| table-row-group | Let the element behave like a <tbody> element |
| table-cell | Let the element behave like a <td> element |
| table-column | Let the element behave like a <col> element |
| table-row | Let the element behave like a <tr> element |
| none | The element is completely removed |
| initial | Sets this property to its default value |
| inherit | Inherits this property from its parent element |

## Display: none;

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

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

## Override The Default Display Value

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

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

A common example is making inline <li> elements for horizontal menus:

li {  
  display: inline;  
}

**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.

## 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:

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:

h1.hidden {  
  visibility: hidden;  
}

## 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

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

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

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

## 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

## 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.

Here is the CSS that is used: 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

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

## position: absolute;

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

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

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

Here is the CSS that is used:

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

## position: sticky;

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

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

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

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

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

## The z-index Property

When elements are positioned, they can overlap other elements.

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

An element can have a positive or negative stack order

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

**Note:** z-index only works on [positioned elements](https://www.w3schools.com/css/css_positioning.asp) (position: absolute, position: relative, position: fixed, or position: sticky) and [flex items](https://www.w3schools.com/css/css3_flexbox.asp) (elements that are direct children of display: flex elements).

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

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

## overflow: hidden

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

div {  
  overflow: hidden;  
}

## overflow: scroll

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

div {  
  overflow: scroll;  
}

## overflow: auto

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

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.

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

## The float Property

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.

img {  
  float: right;  
}

The clear Property

When we use the float property, and we want the next element below (not on right or left), we will have to use the clear property.

The clear property specifies what should happen with the element that is next to a floating element.

The clear property can have one of the following values:

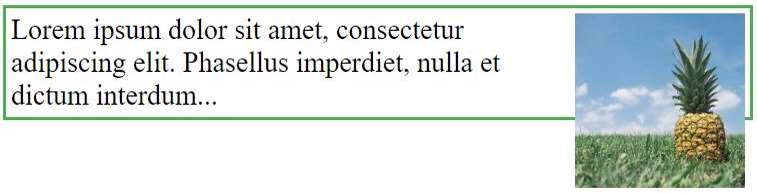
* none - The element is not pushed below left or right floated elements. This is default
* left - The element is pushed below left floated elements
* right - The element is pushed below right floated elements
* both - The element is pushed below both left and right floated elements
* inherit - The element inherits the clear value from its parent

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.

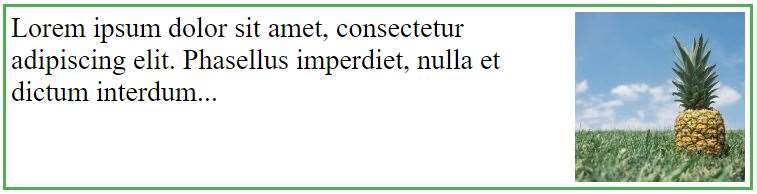
## The clearfix Hack

If a floated element is taller than the containing element, it will "overflow" outside of its container. We can then add a clearfix hack to solve this problem:

### **Without Clearfix**



### **With Clearfix**



.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:

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

**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.

<!DOCTYPE html>

<html>

<head>

<style>

\* {

box-sizing: border-box;

}

.img-container {

float: left;

width: 33.33%;

padding: 5px;

}

.clearfix::after {

content: "";

clear: both;

display: table;

}

</style>

</head>

<body>

<h2>Images Side by Side</h2>

<p>Float images side by side:</p>

<div class="clearfix">

<div class="img-container">

<img src="img\_5terre.jpg" alt="Italy" style="width:100%">

</div>

<div class="img-container">

<img src="img\_forest.jpg" alt="Forest" style="width:100%">

</div>

<div class="img-container">

<img src="img\_mountains.jpg" alt="Mountains" style="width:100%">

</div>

</div>

<p>Note that we also use the clearfix hack to take care of the layout flow, and that we add the box-sizing property to make sure that the image container doesn't break due to extra padding. Try to remove this code to see the effect.</p>

</body>

</html>

## 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:

.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:

Using **Flexbox** to create flexible boxes:

<!DOCTYPE html>

<html>

<head>

<style>

.flex-container {

display: flex;

flex-wrap: nowrap;

background-color: DodgerBlue;

}

.flex-container .box {

background-color: #f1f1f1;

width: 50%;

margin: 10px;

text-align: center;

line-height: 45px;

font-size: 30px;

}

</style>

</head>

<body>

<h1>Flexible Boxes</h1>

<div class="flex-container">

<div class="box">Box 1 - This is some text to make sure that the content gets really tall. This is some text to make sure that the content gets really tall.</div>

<div class="box">Box 2 - My height will follow Box 1.</div>

</div>

<p>Try to resize the browser window to see the flexible layout.</p>

<p><strong>Note:</strong> Flexbox is not supported in Internet Explorer 10 or earlier versions.</p>

</body>

</html>

## 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:

## Center Align Elements

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

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

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

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

## Center Align Text

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

.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

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;:

.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

## 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:

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

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

.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

.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, another solution is to use positioning and the transform property

.center {  
  height: 200px;  
  position: relative;  
  border: 3px solid green;  
}  
  
.center p {  
  margin: 0;  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
}

## Center Vertically - Using Flexbox

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

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

## CSS Combinators

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

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

There are four different combinators in CSS:

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

## Descendant Selector

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

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

div p {  
  background-color: yellow;  
}

## Child Selector (>)

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

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

div > p {  
  background-color: yellow;  
}

## Adjacent Sibling Selector (+)

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

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

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

div + p {  
  background-color: yellow;  
}

## General Sibling Selector (~)

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

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

div ~ p {  
  background-color: yellow;  
}

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

## Hover on <div>

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

div:hover {  
  background-color: blue;  
}

## 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:

<style>

p:first-child {

color: blue;

}

</style>

</head>

<body>

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

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

<div>

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

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

</div>

This is some text.

This is some text.

This is some text.

This is some text.

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

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

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

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

## 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:

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

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

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

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

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

## 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":

<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>

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Example description** |
| [:active](https://www.w3schools.com/cssref/sel_active.asp) | a:active | Selects the active link |
| [:checked](https://www.w3schools.com/cssref/sel_checked.asp) | input:checked | Selects every checked <input> element |
| [:disabled](https://www.w3schools.com/cssref/sel_disabled.asp) | input:disabled | Selects every disabled <input> element |
| [:empty](https://www.w3schools.com/cssref/sel_empty.asp) | p:empty | Selects every <p> element that has no children |
| [:enabled](https://www.w3schools.com/cssref/sel_enabled.asp) | input:enabled | Selects every enabled <input> element |
| [:first-child](https://www.w3schools.com/cssref/sel_firstchild.asp) | p:first-child | Selects every <p> elements that is the first child of its parent |
| [:first-of-type](https://www.w3schools.com/cssref/sel_first-of-type.asp) | p:first-of-type | Selects every <p> element that is the first <p> element of its parent |
| [:focus](https://www.w3schools.com/cssref/sel_focus.asp) | input:focus | Selects the <input> element that has focus |
| [:hover](https://www.w3schools.com/cssref/sel_hover.asp) | a:hover | Selects links on mouse over |
| [:in-range](https://www.w3schools.com/cssref/sel_in-range.asp) | input:in-range | Selects <input> elements with a value within a specified range |
| [:invalid](https://www.w3schools.com/cssref/sel_invalid.asp) | input:invalid | Selects all <input> elements with an invalid value |
| [:lang(*language*)](https://www.w3schools.com/cssref/sel_lang.asp) | p:lang(it) | Selects every <p> element with a lang attribute value starting with "it" |
| [:last-child](https://www.w3schools.com/cssref/sel_last-child.asp) | p:last-child | Selects every <p> elements that is the last child of its parent |
| [:last-of-type](https://www.w3schools.com/cssref/sel_last-of-type.asp) | p:last-of-type | Selects every <p> element that is the last <p> element of its parent |
| [:link](https://www.w3schools.com/cssref/sel_link.asp) | a:link | Selects all unvisited links |
| [:not(selector)](https://www.w3schools.com/cssref/sel_not.asp) | :not(p) | Selects every element that is not a <p> element |
| [:nth-child(n)](https://www.w3schools.com/cssref/sel_nth-child.asp) | p:nth-child(2) | Selects every <p> element that is the second child of its parent |
| [:nth-last-child(n)](https://www.w3schools.com/cssref/sel_nth-last-child.asp) | 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)](https://www.w3schools.com/cssref/sel_nth-last-of-type.asp) | 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)](https://www.w3schools.com/cssref/sel_nth-of-type.asp) | p:nth-of-type(2) | Selects every <p> element that is the second <p> element of its parent |
| [:only-of-type](https://www.w3schools.com/cssref/sel_only-of-type.asp) | p:only-of-type | Selects every <p> element that is the only <p> element of its parent |
| [:only-child](https://www.w3schools.com/cssref/sel_only-child.asp) | p:only-child | Selects every <p> element that is the only child of its parent |
| [:optional](https://www.w3schools.com/cssref/sel_optional.asp) | input:optional | Selects <input> elements with no "required" attribute |
| [:out-of-range](https://www.w3schools.com/cssref/sel_out-of-range.asp) | input:out-of-range | Selects <input> elements with a value outside a specified range |
| [:read-only](https://www.w3schools.com/cssref/sel_read-only.asp) | input:read-only | Selects <input> elements with a "readonly" attribute specified |
| [:read-write](https://www.w3schools.com/cssref/sel_read-write.asp) | input:read-write | Selects <input> elements with no "readonly" attribute |
| [:required](https://www.w3schools.com/cssref/sel_required.asp) | input:required | Selects <input> elements with a "required" attribute specified |
| [:root](https://www.w3schools.com/cssref/sel_root.asp) | root | Selects the document's root element |
| [:target](https://www.w3schools.com/cssref/sel_target.asp) | #news:target | Selects the current active #news element (clicked on a URL containing that anchor name) |
| [:valid](https://www.w3schools.com/cssref/sel_valid.asp) | input:valid | Selects all <input> elements with a valid value |
| [:visited](https://www.w3schools.com/cssref/sel_visited.asp) | a:visited | Selects all visited links |
| Selector | Example | Example description |
| [::after](https://www.w3schools.com/cssref/sel_after.asp) | p::after | Insert content after every <p> element |
| [::before](https://www.w3schools.com/cssref/sel_before.asp) | p::before | Insert content before every <p> element |
| [::first-letter](https://www.w3schools.com/cssref/sel_firstletter.asp) | p::first-letter | Selects the first letter of every <p> element |
| [::first-line](https://www.w3schools.com/cssref/sel_firstline.asp) | p::first-line | Selects the first line of every <p> element |
| [::marker](https://www.w3schools.com/cssref/sel_marker.asp) | ::marker | Selects the markers of list items |
| [::selection](https://www.w3schools.com/cssref/sel_selection.asp) | p::selection | Selects the portion of an element that is selected by a user |

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
* The syntax of pseudo-elements:
* selector::pseudo-element {  
    property: value;  
  }

## The ::first-line Pseudo-element

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

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

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

**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:

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 HTML Classes

Pseudo-elements can be combined with HTML classes:

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

## Multiple Pseudo-elements

Several pseudo-elements can also be combined.

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

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:

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:

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

* Coffee
* Tea
* Milk

1. First
2. Second
3. Third

## CSS - The ::marker Pseudo-element

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

The following example styles the markers of list items:

::marker {  
  color: red;  
  font-size: 23px;  
}

## 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:

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

## Transparent Image

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

img {  
  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:

## 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:

100% opacity

60% opacity

30% opacity

10% opacity

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

## Text in Transparent Box

<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>

# **CSS Navigation Bar**

## 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:

<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:

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

* 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

## Vertical Navigation Bar

To build a vertical navigation bar, you can style the <a> elements inside the list

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

* 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:

## 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:

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;  
}

## 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 from the previous page:

li {  
  display: inline;  
}

* 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:

li {  
  float: left;  
}  
  
a {  
  display: block;  
  padding: 8px;  
  background-color: #dddddd;  
}

* float: left; - Use float to get block elements to float next to each other
* display: block; - Allows us to specify padding (and height, width, margins, etc. if you want)
* padding: 8px; - Specify some padding between each <a> element, 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.

## 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:

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;  
}

### **Right-Align Links**

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

<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:

/\* 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;  
}

### **Fixed Navigation Bar**

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

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

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

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

### **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).

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

**Note:**Internet Explorer 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.

# **CSS Dropdowns**

## Basic Dropdown

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

<style>  
.dropdown {  
  position: relative;  
  display: inline-block;  
}  
  
.dropdown-content {  
  display: none;  
  position: absolute;  
  background-color: #f9f9f9;  
  min-width: 160px;  
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);  
  padding: 12px 16px;  
  z-index: 1;  
}  
  
.dropdown:hover .dropdown-content {  
  display: block;  
}  
</style>  
  
<div class="dropdown">  
  <span>Mouse over me</span>  
  <div class="dropdown-content">  
    <p>Hello World!</p>  
  </div>  
</div>

## Dropdown Menu

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

<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;

.dropdown-content {  
  right: 0;  
}

### **Dropdown Image**

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

<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);

z-index: 1;

}

.dropdown:hover .dropdown-content {

display: block;

}

.desc {

padding: 15px;

text-align: center;

}

</style>

</head>

<body>

<h2>Dropdown Image</h2>

<p>Move the mouse over the image below to open the dropdown content.</p>

<div class="dropdown">

<img src="img\_5terre.jpg" alt="Cinque Terre" width="100" height="50">

<div class="dropdown-content">

<img src="img\_5terre.jpg" alt="Cinque Terre" width="300" height="200">

<div class="desc">Beautiful Cinque Terre</div>

</div>

</div>

## Image Gallery

The following image gallery is created with CSS:

<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>

## 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"):

<style>

#home {

width: 46px;

height: 44px;

background: url(img\_navsprites.gif) 0 0;

}

#next {

width: 43px;

height: 44px;

background: url(img\_navsprites.gif) -91px 0;

}

</style>

</head>

<body>

<img id="home" src="img\_trans.gif" width="1" height="1">

<img id="next" src="img\_trans.gif" width="1" height="1">

</body>

**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:

#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 is 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.

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:

#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: url('img\_navsprites\_hover.gif') 0 -45px;} - For all three hover images we specify the same background position, only 45px further down

## CSS [attribute] Selector

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

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

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

## CSS [attribute="value"] Selector

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

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

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

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

The [attribute~="value"] selector is used to select elements with 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":

[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, whose value can be exactly the specified value, or the specified value followed by a hyphen (-).

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

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

<h2>CSS [attribute|="value"] Selector</h2>

<h1 class="top-header">Welcome</h1>

<p class="top-text">Hello world!</p>

<p class="topcontent">Are you learning CSS?</p>

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

The [attribute^="value"] selector is used to select elements with the specified attribute, whose value starts with the specified value.

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

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

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

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

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

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

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

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

## 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!

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

## Styling Input Fields

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

input {  
  width: 100%;  
}

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

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

## Padded Inputs

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

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

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

## Bordered Inputs

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

input[type=text] {  
  border: 2px solid red;  
  border-radius: 4px;  
}

If you only want a bottom border, use the border-bottom property: 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:

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:

input[type=text]:focus {  
  background-color: lightblue;  
}

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

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](https://www.w3schools.com/css/css3_transitions.asp) chapter.

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):

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

select {  
  width: 100%;  
  padding: 16px 20px;  
  border: none;  
  border-radius: 4px;  
  background-color: #f1f1f1;  
}

<form>

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

<option value="au">Australia</option>

<option value="ca">Canada</option>

<option value="usa">USA</option>

</select>

</form>

## Styling Input Buttons

input[type=button], input[type=submit], input[type=reset] {  
  background-color: #04AA6D;  
  border: none;  
  color: white;  
  padding: 16px 32px;  
  text-decoration: none;  
  margin: 4px 2px;  
  cursor: pointer;  
}  
  
/\* Tip: use **width: 100%** for full-width buttons \*/

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:

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>":

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) " ";  
}

## 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:

.header {  
  background-color: #F1F1F1;  
  text-align: center;  
  padding: 20px;  
}

## Navigation Bar

A navigation bar contains a list of links to help visitors navigating through your website:

/\* 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;  
}

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

/\* Create three equal columns that float 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%;  
  }  
}

**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](https://www.w3schools.com/css/css3_mediaqueries.asp).

**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](https://www.w3schools.com/css/css3_flexbox.asp).

## 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:

.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%;  
  }  
}

## Footer

The footer is placed at the bottom of your page. It often contains information like copyright and contact info:

.footer {  
  background-color: #F1F1F1;  
  text-align: center;  
  padding: 10px;  
}

## 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 |
| mm | millimeters |
| in | inches (1in = 96px = 2.54cm) |
| px \* | pixels (1px = 1/96th of 1in) |
| pt | points (1pt = 1/72 of 1in) |
| pc | picas (1pc = 12 pt) |

\* 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 scale 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) |
| ex | Relative to the x-height of the current font (rarely used) |
| ch | Relative to width of the "0" (zero) |
| rem | Relative to font-size of the root element |
| vw | Relative to 1% of the width of the viewport\* |
| vh | Relative to 1% of the height of the viewport\* |
| vmin | Relative to 1% of viewport's\* smaller dimension |
| vmax | Relative to 1% of viewport's\* larger dimension |
| % | Relative to the parent element |

**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 | Chrome | Edge | Firefox | Safari | Opera |
| 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 | 12.0 | 19.0 | 6.0 | 20.0 |
| vmax | 26.0 | 16.0 | 19.0 | 7.0 | 20.0 |

## What is Specificity?

If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element.

Think of specificity as a score/rank that determines which style declaration is ultimately applied to an element.

In this example, we have added a class selector (named "test"), and specified a green color for this class. The text will now be green (even though we have specified a red color for the element selector "p"). This is because the class selector is given higher priority:

<html>  
<head>  
  <style>  
    .test {color: green;}  
    p {color: red;}  
  </style>  
</head>  
<body>  
  
<p class="test">Hello World!</p>  
  
</body>  
</html>

In this example, we have added a class selector (named "test"), and specified a green color for this class. The text will now be green (even though we have specified a red color for the element selector "p"). This is because the class selector is given higher priority:

<html>  
<head>  
  <style>  
    .test {color: green;}  
    p {color: red;}  
  </style>  
</head>  
<body>  
  
<p class="test">Hello World!</p>

In this example, we have added the id selector (named "demo"). The text will now be blue, because the id selector is given higher priority:

<html>  
<head>  
  <style>  
    #demo {color: blue;}  
    .test {color: green;}  
    p {color: red;}  
  </style>  
</head>  
<body>  
  
<p id="demo" class="test">Hello World!</p>

In this example, we have added an inline style for the "p" element. The text will now be pink, because the inline style is given the highest priority:

<html>  
<head>  
  <style>  
    #demo {color: blue;}  
    .test {color: green;}  
    p {color: red;}  
  </style>  
</head>  
<body>  
  
<p id="demo" class="test" style="color: pink;">Hello World!</p>

Specificity Hierarchy

Every CSS selector has its place in the specificity hierarchy.

There are four categories which define the specificity level of a selector:

1. **Inline styles** - Example: <h1 style="color: pink;">
2. **IDs** - Example: #navbar
3. **Classes, pseudo-classes, attribute selectors** - Example: .test, :hover, [href]
4. **Elements and pseudo-elements** - Example: h1, ::before

## How to Calculate Specificity?

Memorize how to calculate specificity!

Start at 0, add 100 for each ID value, add 10 for each class value (or pseudo-class or attribute selector), add 1 for each element selector or pseudo-element.

**Note:** Inline style gets a specificity value of 1000, and is always given the highest priority!

**Note 2:** There is one exception to this rule: if you use the [!important](https://www.w3schools.com/css/css_important.asp) rule, it will even override inline styles!

|  |  |  |
| --- | --- | --- |
| **Selector** | **Specificity Value** | **Calculation** |
| p | 1 | 1 |
| p.test | 11 | 1 + 10 |
| p#demo | 101 | 1 + 100 |
| <p style="color: pink;"> | 1000 | 1000 |
| #demo | 100 | 100 |
| .test | 10 | 10 |
| p.test1.test2 | 21 | 1 + 10 + 10 |
| #navbar p#demo | 201 | 100 + 1 + 100 |
| \* | 0 | 0 (the universal selector is ignored) |

**The selector with the highest specificity value will win and take effect!**

Consider these three code fragments:

A: h1  
B: h1#content  
C: <h1 id="content" style="color: pink;">Heading</h1>

The specificity of A is 1 (one element selector)  
The specificity of B is 101 (one ID reference + one element selector)  
The specificity of C is 1000 (inline styling)

Since the third rule (C) has the highest specificity value (1000), this style declaration will be applied.

## More Specificity Rules Examples

**Equal specificity: the latest rule wins**- If the same rule is written twice into the external style sheet, then the latest rule wins:

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

**ID selectors have a higher specificity than attribute selectors** - Look at the following three code lines:

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

This is a div

the first rule is more specific than the other two, and will therefore 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/\*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:

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

**The universal selector (\*) and inherited values have a specificity of 0** - The universal selector (\*) and inherited values are ignored!

## What is !important?

The !important rule in CSS is used to add more importance to a property/value than normal.

In fact, if you use the !important rule, it will override ALL previous styling rules for that specific property on that element!

#myid {  
  background-color: blue;  
}  
  
.myclass {  
  background-color: gray;  
}  
  
p {  
  background-color: red **!important**;  
}

This is some text in a paragraph.

In the example above. all three paragraphs will get a red background color, even though the ID selector and the class selector have a higher specificity. The !important rule overrides the background-color property in both cases.

## Important About !important

The only way to override an !important rule is to include another !important rule on a declaration with the same (or higher) specificity in the source code - and here the problem starts! This makes the CSS code confusing and the debugging will be hard, especially if you have a large style sheet!

Here we have created a simple example. It is not very clear, when you look at the CSS source code, which color is considered most important:

#myid {  
  background-color: blue **!important**;  
}  
  
.myclass {  
  background-color: gray **!important**;  
}  
  
p {  
  background-color: red **!important**;  
}

**Tip:** It is good to know about the !important rule. You might see it in some CSS source code. However, do not use it unless you absolutely have to.

## Maybe One or Two Fair Uses of !important

One way to use !important is if you have to override a style that cannot be overridden in any other way. This could be if you are working on a Content Management System (CMS) and cannot edit the CSS code. Then you can set some custom styles to override some of the CMS styles.

Another way to use !important is: Assume you want a special look for all buttons on a page. Here, buttons are styled with a gray background color, white text, and some padding and border:

.button {  
  background-color: #8c8c8c;  
  color: white;  
  padding: 5px;  
  border: 1px solid black;  
}

The look of a button can sometimes change if we put it inside another element with higher specificity, and the properties get in conflict. Here is an example of this:

.button {  
  background-color: #8c8c8c;  
  color: white;  
  padding: 5px;  
  border: 1px solid black;  
}  
  
#myDiv a {  
  color: red;  
  background-color: yellow;  
}

# **CSS Math Functions**

## The calc() Function

The calc() function performs a calculation to be used as the property value.

### **CSS Syntax**

calc(expression)

|  |  |
| --- | --- |
| **Value** | **Description** |
| *expression* | Required. A mathematical expression. The result will be used as the value. The following operators can be used: + - \* / |

Use calc() to calculate the width of a <div> element:

#div1 {  
  position: absolute;  
  left: 50px;  
  width: calc(100% - 100px);  
  border: 1px solid black;  
  background-color: yellow;  
  padding: 5px;  
}

## The max() Function

The max() function uses the largest value, from a comma-separated list of values, as the property value.

### **CSS Syntax**

max(value1, value2, ...)

|  |  |
| --- | --- |
| **Value** | **Description** |
| value1, value2, ... | Required. A list of comma-separated values - where the largest value is chosen |

Use max() to set the width of #div1 to whichever value is largest, 50% or 300px:

#div1 {  
  background-color: yellow;  
  height: 100px;  
  width: max(50%, 300px);  
}

## The min() Function

The min() function uses the smallest value, from a comma-separated list of values, as the property value.

### **CSS Syntax**

min(value1, value2, ...)

|  |  |
| --- | --- |
| **Value** | **Description** |
| value1, value2, ... | Required. A list of comma-separated values - where the smallest value is chosen |

Use min() to set the width of #div1 to whichever value is smallest, 50% or 300px:

#div1 {  
  background-color: yellow;  
  height: 100px;  
  width: min(50%, 300px);  
}

## 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!

#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

You could also create elliptical corners: #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 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!

#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

#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.

Different slice values completely changes the look of the border:

#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 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):

#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):

#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.

#div1 {  
  background: url(img\_flower.jpg);  
  background-size: 100px 80px;  
  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:

#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:

html {  
  background: url(img\_man.jpg) no-repeat center fixed;  
  background-size: cover;  
}

## Hero Image

You could also use different background properties on a <div> to create a hero image (a large image with text), and place it where you want.

.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

#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:

#example1 {  
  border: 10px dotted black;  
  padding: 35px;  
  background: yellow;  
  background-clip: content-box;  
}

## 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).

#p1 {background-color: rgba(255, 0, 0, 0.3);}  /\* red 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.

#p1 {background-color: hsl(120, 100%, 50%);}  /\* 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).

#p1 {background-color: hsla(120, 100%, 50%, 0.3);}  /\* 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).

#p1 {background-color:rgb(255,0,0);opacity:0.6;}  /\* red with opacity \*/

## The transparent Keyword

The transparent keyword is used to make a color transparent. This is often used to make a transparent background color for an element.

Here, the background color of the <div> element will be fully transparent, and the background image will show through:

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

## The currentcolor Keyword

The currentcolor keyword is like a variable that holds the current value of the color property of an element.

This keyword can be useful if you want a specific color to be consistent in an element or a page.

In this example the border color of the <div> element will be blue, because the text color of the <div> element is blue:

div {  
  color: blue;  
  border: 10px solid currentcolor;  
}

In this example the <div>'s background color is set to the current color value of the body element:

body {  
  color: purple;  
}  
  
div {  
  background-color: currentcolor;  
}

In this example the <div>'s border color and shadow color is set to the current color value of the body element:

body {  
 color: green;  
}  
  
div {  
  box-shadow: 0px 0px 15px currentcolor;  
  border: 5px solid currentcolor;  
}

## The inherit Keyword

The inherit keyword specifies that a property should inherit its value from its parent element.

The inherit keyword can be used for any CSS property, and on any HTML element.

div {  
  border: 2px solid red;  
}  
  
span {  
  border: inherit;  
}

# **CSS Gradients**

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

CSS defines three types of gradients:

* **Linear Gradients (goes down/up/left/right/diagonally)**
* **Radial Gradients (defined by their center)**
* **Conic Gradients (rotated around a center point)**

## 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, ...);

**Direction - Top to Bottom (this is default)**

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

**Direction - Left to Right**

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

**Direction – Diagonal**

#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.). A value of 0deg is equivalent to "to top". A value of 90deg is equivalent to "to right". A value of 180deg is equivalent to "to bottom".

#grad {  
  background-image: linear-gradient(180deg, red, yellow);  
}

## Using Multiple Color Stops

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

## 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).

#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:

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);

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

#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:

#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:

#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**
* 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:

A repeating radial gradient:

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

## CSS Conic Gradients

A conic gradient is a gradient with color transitions rotated around a center point.

To create a conic gradient you must define at least two colors.

### **Syntax**

background-image: conic-gradient([from angle] [at position,] color [degree], color [degree], ...);

By default, angle is 0deg and position is center.

If no degree is specified, the colors will be spread equally around the center point.

## Conic Gradient: Three Colors

A conic gradient with three colors:

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

## Conic Gradient: Three Colors and Degrees

A conic gradient with three colors and a degree for each color:

#grad {  
  background-image: conic-gradient(red 45deg, yellow 90deg, green 210deg);  
}

## Create Pie Charts

Just add border-radius: 50% to make the conic gradient look like a pie:

#grad {  
  background-image: conic-gradient(red 0deg, red 90deg, yellow 90deg, yellow 180deg, green 180deg, green 270deg, blue 270deg);  
  border-radius: 50%;  
}

## Conic Gradient With Specified From Angle

The [from angle] specifies an angle that the entire conic gradient is rotated by.

The following example shows a conic gradient with a from angle of 90deg:

#grad {  
  background-image: conic-gradient(from 90deg, red, yellow, green);  
}

## Conic Gradient With Specified Center Position

The [at position] specifies the center of the conic gradient.

The following example shows a conic gradient with a center position of 60% 45%:

A conic gradient with a specified center position:

#grad {  
  background-image: conic-gradient(at 60% 45%, red, yellow, green);  
}

## Repeating a Conic Gradient

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

#grad {  
  background-image: repeating-conic-gradient(red 10%, yellow 20%);  
  border-radius: 50%;  
}

CSS Shadow Effects

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

In these chapters 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):

h1 {  
  text-shadow: 2px 2px;  
}

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

Then, add a blur effect to the shadow:

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

The following example shows a white text with black shadow:

h1 {  
  color: white;  
  text-shadow: 2px 2px 4px #000000;  
}

The following example shows a red neon glow shadow:

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:

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

The following example shows a white text with black, blue, and darkblue shadow:

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)

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

## CSS box-shadow Property

The CSS box-shadow property is used to apply one or more shadows to an element.

## Specify a Horizontal and a Vertical Shadow

In its simplest use, you only specify a horizontal and a vertical shadow. The default color of the shadow is the current text-color.

Specify a horizontal and a vertical shadow:

div {  
  box-shadow: 10px 10px;  
}

Specify a color for the shadow:

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

Add a blur effect to the shadow:

div {  
  box-shadow: 10px 10px 5px lightblue;  
}

## Set the Spread Radius of the Shadow

The spread parameter defines the spread radius. A positive value increases the size of the shadow, a negative value decreases the size of the shadow.

Set the spread radius of the shadow:

div {  
  box-shadow: 10px 10px 5px 12px lightblue;  
}

## Set the inset Parameter

The inset parameter changes the shadow from an outer shadow (outset) to an inner shadow.

Add the inset parameter:

div {  
  box-shadow: 10px 10px 5px lightblue inset;  
}

## Add Multiple Shadows

An element can also have multiple shadows:

div {  
  box-shadow: 5px 5px blue, 10px 10px red, 15px 15px green;  
}

## Cards

You can also use the box-shadow property to create paper-like cards:

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 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
* The CSS text-overflow property specifies how overflowed content that is not displayed should be signaled to the user.
* It can be clipped
* or it can be rendered as an ellipsis (...):

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;  
}

## 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

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.

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.

p.test1 {  
  writing-mode: horizontal-tb;  
}  
  
span.test2 {  
  writing-mode: vertical-rl;  
}  
  
p.test2 {  
  writing-mode: vertical-rl;  
}

## 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.

## 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:

@font-face {  
  font-family: myFirstFont;  
  src: url(sansation\_light.woff);  
}  
  
div {  
  font-family: myFirstFont;  
}

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()

## 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).

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

## The rotate() Method

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

div {  
  transform: rotate(20deg);  
}

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

## The scaleX() Method

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

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

div {  
  transform: scaleX(2);  
}

div {  
  transform: scaleX(0.5);  
}

## The scaleY() Method

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

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

div {  
  transform: scaleY(3);  
}

div {  
  transform: scaleY(0.5);  
}

## The skewX() Method

The skewX() method skews an element along the X-axis by the given angle.

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 <div> element 20 degrees along the Y-axis:

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

div {  
  transform: skew(20deg, 10deg);  
}

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

div {  
  transform: matrix(1, -0.3, 0, 1, 0, 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:

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

## The rotateY() Method

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

#myDiv {  
  transform: rotateY(150deg);  
}

## The rotateZ() Method

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

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

## CSS Transitions

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

In this chapter you will learn about the following properties:

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

How to Use CSS Transitions?

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

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

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

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

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

## Change Several Property Values

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

div {  
  transition: width 2s, height 4s;  
}

Specify the Speed Curve of the Transition

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

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

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

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

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

div {  
  transition-property: width;  
  transition-duration: 2s;  
  transition-timing-function: linear;  
  transition-delay: 1s;  
}

or by using the shorthand property transition:

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

## CSS Animations

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

## The @keyframes Rule

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

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

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

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

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

/\* 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:

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:

ackground-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:

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

Run Animation in Reverse Direction or Alternate Cycles

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

The animation-direction property can have the following values:

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

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

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

Specify the Speed Curve of the Animation

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

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

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

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

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

Specify the fill-mode For an Animation

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

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

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

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

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

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

## Animation Shorthand Property

The example below uses six of the animation properties:

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

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

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

## Basic Tooltip

Create a tooltip that appears when the user moves the mouse over an element:

<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>

## Positioning Tooltips

In this example, the tooltip is placed to the right (left:105%) of the "hoverable" text (<div>). 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%;  
}

### **Left Tooltip**

.tooltip .tooltiptext {  
  top: -5px;  
  right: 105%;  
}

### **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 \*/  
}

### **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 \*/  
}

## 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:

.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;  
}

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;  
}

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;  
}

## 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;  
}

## Rounded Images

Use the border-radius property to create rounded images:

Rounded Image:

img {  
  border-radius: 8px;  
}

Circled Image:

img {  
  border-radius: 50%;  
}

## Thumbnail Images

Use the border property to create thumbnail images.

### **Example**

img {  
  border: 1px solid #ddd;  
  border-radius: 4px;  
  padding: 5px;  
  width: 150px;  
}  
  
<img src="paris.jpg" alt="Paris">

### **Example**

<style>

img {

border: 1px solid #ddd;

border-radius: 4px;

padding: 5px;

width: 150px;

}

img:hover {

box-shadow: 0 0 2px 1px rgba(0, 140, 186, 0.5);

}

</style>

</head>

<body>

<h2>Thumbnail Image as Link</h2>

<p>Use the border property to create thumbnail images. Wrap an anchor around the image to use it as a link.</p>

<p>Hover over the image and click on it to see the effect.</p>

<a target="\_blank" href="paris.jpg">

<img src="paris.jpg" alt="Paris" style="width:150px">

</a>

</body>

## Responsive Images

Responsive images will automatically adjust to fit the size of the screen.

### **Example**

img {  
  max-width: 100%;  
  height: auto;  
}

## Center an Image

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

img {  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
  width: 50%;  
}

## Image Filters

The CSS filter property adds visual effects (like blur and saturation) to an element.

Change the color of all images to black and white (100% gray):

img {  
  filter: grayscale(100%);  
}

.blur {filter: blur(4px);}

.brightness {filter: brightness(250%);}

.contrast {filter: contrast(180%);}

.grayscale {filter: grayscale(100%);}

.huerotate {filter: hue-rotate(180deg);}

.invert {filter: invert(100%);}

.opacity {filter: opacity(50%);}

.saturate {filter: saturate(7);}

.sepia {filter: sepia(100%);}

.shadow {filter: drop-shadow(8px 8px 10px green);}

## Flip an Image

### **Example**

img:hover {  
  transform: scaleX(-1);  
}

## CSS Image Reflections

The box-reflect property is used to create an image reflection.

The value of the box-reflect property can be: below, above, left , or right.

Here we want the reflection to the right of the image:

img {  
  -webkit-box-reflect: right;  
}

## CSS Reflection Offset

To specify the gap between the image and the reflection, add the size of the gap to the box-reflect property.

Here we want the reflection below the image, with a 20px offset:

img {  
  -webkit-box-reflect: below 20px;  
}

## CSS Reflection Offset

To specify the gap between the image and the reflection, add the size of the gap to the box-reflect property.

Here we want the reflection below the image, with a 20px offset:

img {  
  -webkit-box-reflect: below 20px;  
}

## CSS Reflection With Gradient

We can also create a fade-out effect on the reflection.

Create a fade-out effect on the reflection:

img {  
  -webkit-box-reflect: below 0px linear-gradient(to bottom, rgba(0,0,0,0.0), rgba(0,0,0,0.4));  
}

## 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".

We see that the image is being squished to fit the container of 200x300 pixels (its original aspect ratio is destroyed).

Here is where the object-fit property comes in. The object-fit property can take one of the following values:

* fill - This is default. The image is resized to fill the given dimension. If necessary, the image will be stretched or squished to fit
* contain - The image keeps its aspect ratio, but is resized to fit within the given dimension
* cover - The image keeps its aspect ratio and fills the given dimension. The image will be clipped to fit
* none - The image is not resized
* scale-down - the image is scaled down to the smallest version of none or contain

## Using object-fit: cover;

If we use object-fit: cover; the image keeps its aspect ratio and fills the given dimension. The image will be clipped to fit:

img {  
  width: 200px;  
  height: 300px;  
  object-fit: cover;  
}

## Using object-fit: contain;

If we use object-fit: contain; the image keeps its aspect ratio, but is resized to fit within the given dimension

## Using object-fit: fill;

If we use object-fit: fill; the image is resized to fill the given dimension. If necessary, the image will be stretched or squished to fit

## Using object-fit: none;

If we use object-fit: none; the image is not resized

## Using object-fit: scale-down;

If we use object-fit: scale-down; the image is scaled down to the smallest version of none or contain.

## CSS object-fit More Examples

.fill {object-fit: fill;}  
.contain {object-fit: contain;}  
.cover {object-fit: cover;}  
.scale-down {object-fit: scale-down;}  
.none {object-fit: none;}

## Using the object-position Property

Let's say that the part of the image that is shown, is not positioned as we want. To position the image, we will use the object-position property.

img {  
  width: 200px;  
  height: 300px;  
  object-fit: cover;  
  object-position: 80% 100%;  
}

## The CSS mask-image Property

The CSS mask-image property specifies a mask layer image.

The mask layer image can be a PNG image, an SVG image, a [CSS gradient](https://www.w3schools.com/css/css3_gradients.asp), or an [SVG <mask> element](https://www.w3schools.com/graphics/svg_intro.asp).

With CSS masking you create a mask layer to place over an element to partially or fully hide portions of the element.

## Use an Image as the Mask Layer

To use a PNG or an SVG image as the mask layer, use a url() value to pass in the mask layer image.

The mask image needs to have a transparent or semi-transparent area. Black indicates fully transparent.

Here is the source code:

.mask1 {  
  -webkit-mask-image: url(w3logo.png);  
  mask-image: url(w3logo.png);  
  -webkit-mask-repeat: no-repeat;  
  mask-repeat: no-repeat;  
}

<div class="mask1">

<img src="img\_5terre.jpg" alt="Cinque Terre" width="600" height="400">

</div>

If we omit the mask-repeat property, the mask image will be repeated all over the image from Cinque Terre, Italy.

## Use Gradients as the Mask Layer

CSS linear and radial gradients can also be used as mask images.

Use a linear gradient as a mask layer:

.mask1 {  
  -webkit-mask-image: linear-gradient(black, transparent);  
  mask-image: linear-gradient(black, transparent);  
}

### **Radial Gradient Examples**

Here, we use a radial-gradient (shaped as a circle) as the mask layer for our image

Use a radial gradient as a mask layer (a circle):

.mask2 {  
  -webkit-mask-image: radial-gradient(circle, black 50%, rgba(0, 0, 0, 0.5) 50%);  
  mask-image: radial-gradient(circle, black 50%, rgba(0, 0, 0, 0.5) 50%);  
}

Use another radial gradient as a mask layer (an ellipse):

.mask3 {  
  -webkit-mask-image: radial-gradient(ellipse, black 50%, rgba(0, 0, 0, 0.5) 50%);  
  mask-image: radial-gradient(ellipse, black 50%, rgba(0, 0, 0, 0.5) 50%);  
}

An SVG mask layer (formed as a triangle):

<svg width="600" height="400">  
  <mask id="svgmask1">  
    <polygon fill="#ffffff" points="200 0, 400 400, 0 400"></polygon>  
  </mask>  
  <image xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="img\_5terre.jpg" mask="url(#svgmask1)"></image>  
</svg>

An SVG mask layer (formed as a star):

<svg width="600" height="400">  
  <mask id="svgmask2">  
    <polygon fill="#ffffff" points="100,10 40,198 190,78 10,78 160,198"></polygon>  
  </mask>  
  <image xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="img\_5terre.jpg" mask="url(#svgmask2)"></image>  
</svg>

An SVG mask layer (formed as circles):

<svg width="600" height="400">  
  <mask id="svgmask3">  
    <circle fill="#ffffff" cx="75" cy="75" r="75"></circle>  
    <circle fill="#ffffff" cx="80" cy="260" r="75"></circle>  
    <circle fill="#ffffff" cx="270" cy="160" r="75"></circle>  
  </mask>  
  <image xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="img\_5terre.jpg" mask="url(#svgmask3)"></image>  
</svg>

## Basic Button Styling

.button {  
  background-color: #04AA6D; /\* Green \*/  
  border: none;  
  color: white;  
  padding: 15px 32px;  
  text-align: center;  
  text-decoration: none;  
  display: inline-block;  
  font-size: 16px;  
}

## Button Colors

.button1 {background-color: #04AA6D;} /\* 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

.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:

.button1 {padding: 10px 24px;}  
.button2 {padding: 12px 28px;}  
.button3 {padding: 14px 40px;}

## Rounded Buttons

Use the border-radius property to add rounded corners to a button:

.button1 {border-radius: 2px;}  
.button2 {border-radius: 4px;}  
.button3 {border-radius: 8px;}

## Colored Button Borders

.button1 {  
  background-color: white;  
  color: black;  
  border: 2px solid #04AA6D; /\* 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:

.button {  
  transition-duration: 0.4s;  
}  
  
.button:hover {  
  background-color: #04AA6D; /\* Green \*/  
  color: white;  
}  
...

## Shadow Buttons

.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:

.disabled {  
  opacity: 0.6;  
  cursor: not-allowed;  
}

## Button Width

.button1 {width: 250px;}  
.button2 {width: 50%;}  
.button3 {width: 100%;}

## Button Groups

.button {  
  float: left;  
}

Use the border property to create a bordered button group:

.button {  
  float: left;  
  border: 1px solid green;  
}

Use display:block instead of float:left to group the buttons below each other, instead of side by side:

.button {  
  display: block;  
}

## Button on Image

<style>

.container {

position: relative;

width: 100%;

max-width: 400px;

}

.container img {

width: 100%;

height: auto;

}

.container .btn {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

-ms-transform: translate(-50%, -50%);

background-color: #f1f1f1;

color: black;

font-size: 16px;

padding: 16px 30px;

border: none;

cursor: pointer;

border-radius: 5px;

text-align: center;

}

.container .btn:hover {

background-color: black;

color: white;

}

</style>

</head>

<body>

<h2>Button on Image</h2>

<p>Add a button on an image:</p>

<div class="container">

<img src="img\_lights.jpg" alt="Snow" style="width:100%">

<button class="btn">Button</button>

</div>

# **CSS Pagination**

If you have a website with lots of pages, you may wish to add some sort of pagination to each page:

.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:

.pagination a.active {  
  background-color: #4CAF50;  
  color: white;  
}  
  
.pagination a:hover:not(.active) {background-color: #ddd;}

### **Rounded Active and Hoverable Buttons**

.pagination a {  
  border-radius: 5px;  
}  
  
.pagination a.active {  
  border-radius: 5px;  
}

### **Hoverable Transition Effect**

.pagination a {  
  transition: background-color .3s;  
}

## Bordered Pagination

.pagination a {  
  border: 1px solid #ddd; /\* Gray \*/  
}

### **Rounded Borders**

**Tip:** Add rounded borders to your first and last link in the pagination:

.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:

.pagination a {  
  margin: 0 4px; /\* 0 is for top and bottom. Feel free to change it \*/  
}

## Breadcrumbs

Another variation of pagination is so-called "breadcrumbs":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 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

## 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 <div> element into 3 columns:

div {  
  column-count: 3;  
}

## CSS Specify the Gap Between Columns

The column-gap property specifies the gap between the columns.

div {  
  column-gap: 40px;  
}

## CSS Column Rules

div {  
  column-rule-style: solid;  
}

div {  
  column-rule-width: 1px;  
}

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:

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:

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:

div {  
  column-width: 100px;  
}

CSS User Interface

In this chapter you will learn about the following CSS user interface properties:

* resize
* outline-offset

## CSS Resizing

The resize property specifies if (and how) an element should be resizable by the user.

div {  
  resize: horizontal;  
  overflow: auto;  
}

div {  
  resize: vertical;  
  overflow: auto;  
}

div {  
  resize: both;  
  overflow: auto;  
}

In many browsers, <textarea> is resizable by default. Here, we have used the resize property to disable the resizability:

<style>

textarea#test {

resize: none;

}

</style>

</head>

<body>

<h1>The resize Property</h1>

<p>In many browsers, textarea elements are resizable by default. In this example, we have used the resize property to disable the resizability:</p>

<textarea id="test">Textarea - Not resizable</textarea>

<br><br>

<textarea>Textarea - Resizable (default)</textarea>

## CSS Outline Offset

The outline-offset property adds space between an outline and the edge or border of an element.

**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.

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 Variables

The var() function is used to insert the value of a CSS variable.

CSS variables have access to the DOM, which means that you can create variables with local or global scope, change the variables with JavaScript, and change the variables based on media queries.

A good way to use CSS variables is when it comes to the colors of your design. Instead of copy and paste the same colors over and over again, you can place them in variables.

Syntax of the var() Function

The var() function is used to insert the value of a CSS variable.

The syntax of the var() function is as follows:

var(--*name, value*)

|  |  |
| --- | --- |
| **Value** | **Description** |
| *name* | Required. The variable name (must start with two dashes) |
| *value* | Optional. The fallback value (used if the variable is not found) |

**Note:** The variable name must begin with two dashes (--) and it is case sensitive!

## How var() Works

First of all: CSS variables can have a global or local scope.

Global variables can be accessed/used through the entire document, while local variables can be used only inside the selector where it is declared.

To create a variable with global scope, declare it inside the :root selector. The [:root](https://www.w3schools.com/cssref/sel_root.asp) selector matches the document's root element.

To create a variable with local scope, declare it inside the selector that is going to use it.

First, we declare two global variables (--blue and --white). Then, we use the var() function to insert the value of the variables later in the style sheet:

:root {  
  --blue: #1e90ff;  
  --white: #ffffff;  
}  
  
body { background-color: var(--blue); }  
  
h2 { border-bottom: 2px solid var(--blue); }  
  
.container {  
  color: var(--blue);  
  background-color: var(--white);  
  padding: 15px;  
}  
  
button {  
  background-color: var(--white);  
  color: var(--blue);  
  border: 1px solid var(--blue);  
  padding: 5px;  
}

Advantages of using var() are:

* makes the code easier to read (more understandable)
* makes it much easier to change the color values

## Override Global Variable With Local Variable

From the previous page we have learned that global variables can be accessed/used through the entire document, while local variables can be used only inside the selector where it is declared.

:root {  
  --blue: #1e90ff;  
  --white: #ffffff;  
}  
  
body {  
  background-color: var(--blue);  
}  
  
h2 {  
  border-bottom: 2px solid var(--blue);  
}  
  
.container {  
  color: var(--blue);  
  background-color: var(--white);  
  padding: 15px;  
}  
  
button {  
  --blue: #0000ff; /\* local variable will override global \*/  
  background-color: var(--white);  
  color: var(--blue);  
  border: 1px solid var(--blue);  
  padding: 5px;  
}

## Using Variables in Media Queries

Now we want to change a variable value inside a media query.

Here, we first declare a new local variable named --fontsize for the .container class. We set its value to 25 pixels. Then we use it in the .container class further down. Then, we create a @media rule that says "When the browser's width is 450px or wider, change the --fontsize variable value of the .container class to 50px."

/\* Variable declarations \*/  
:root {  
  --blue: #1e90ff;  
  --white: #ffffff;  
}  
  
.container {  
  --fontsize: 25px;  
}  
  
/\* Styles \*/  
body {  
  background-color: var(--blue);  
}  
  
h2 {  
  border-bottom: 2px solid var(--blue);  
}  
  
.container {  
  color: var(--blue);  
  background-color: var(--white);  
  padding: 15px;  
  font-size: var(--fontsize);  
}  
  
@media screen and (min-width: 450px) {  
  .container {  
    --fontsize: 50px;  
  }  
}

## 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 <div> elements with the same specified width and height:

The two <div> elements above end up with different sizes in the result (because div2 has a padding specified):

.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

.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:

\* {  
  box-sizing: border-box;  
}

SS Media Queries

The @media rule, introduced in CSS2, made it possible to define different style rules for different media types.

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
* orientation of the viewport (landscape or portrait)
* 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).

CSS Media Types

|  |  |
| --- | --- |
| **Value** | **Description** |
| all | Used for all media type devices |
| print | Used for print preview mode |
| screen | Used for computer screens, tablets, smart-phones etc. |

CSS Common Media Features

Here are some commonly used media features:

|  |  |
| --- | --- |
| **Value** | **Description** |
| orientation | Orientation of the viewport. Landscape or portrait |
| max-height | Maximum height of the viewport |
| min-height | Minimum height of the viewport |
| height | Height of the viewport (including scrollbar) |
| max-width | Maximum width of the viewport |
| min-width | Minimum width of the viewport |
| width | Width of the viewport (including scrollbar) |

Media Query Syntax

A media query consists of a media type and can contain one or more media features, which resolve to either true or false.

@media not|only *mediatype*and(*media feature*) and(*media feature*) { *CSS-Code;*}

The *mediatype* is optional (if omitted, it will be set to all). However, if you use **not** or **only**, you must also specify a *mediatype*.

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 media features 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.

Meaning of the **not**, **only,** and **and** keywords:

**not:** This keyword inverts the meaning of an entire media query.

**only:** This keyword prevents older browsers that do not support media queries from applying the specified styles. **It has no effect on modern browsers.**

**and:** This keyword combines a media type and one or more media features.

You can also link to different stylesheets for different media and different widths of the browser window (viewport):

<link rel="stylesheet" media="print" href="print.css">  
<link rel="stylesheet" media="screen" href="screen.css">  
<link rel="stylesheet" media="screen and (min-width: 480px)" href="example1.css">  
<link rel="stylesheet" media="screen and (min-width: 701px) and (max-width: 900px)" href="example2.css">  
etc....

## 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):

@media screen and (min-width: 480px) {  
  body {  
    background-color: lightgreen;  
  }  
}

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](https://www.w3schools.com/css/css_rwd_intro.asp).

## 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:

/\* 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;  
  }  
}

Float:

/\* 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:

/\* 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:

/\* 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;  
  }  
}

## 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:

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 <div> element:

@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:

/\* 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;  
  }  
}

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

## Flexbox Elements

To start using the Flexbox model, you need to first define a flex container.

A flex container with three flex items:

<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div>3</div>  
</div>

## Parent Element (Container)

Like we specified in the previous chapter, this is a flex **container**(the blue area) with three flex **items**:

.flex-container {  
  display: flex;  
}

The flex container properties are:

* [flex-direction](https://www.w3schools.com/css/css3_flexbox_container.asp#flex-direction)
* [flex-wrap](https://www.w3schools.com/css/css3_flexbox_container.asp#flex-wrap)
* [flex-flow](https://www.w3schools.com/css/css3_flexbox_container.asp#flex-flow)
* [justify-content](https://www.w3schools.com/css/css3_flexbox_container.asp#justify-content)
* [align-items](https://www.w3schools.com/css/css3_flexbox_container.asp#align-items)
* [align-content](https://www.w3schools.com/css/css3_flexbox_container.asp#align-content)

## The flex-direction Property

The flex-direction property defines in which direction the container wants to stack the flex items.

The column value stacks the flex items vertically (from top to bottom):

.flex-container {  
  display: flex;  
  flex-direction: column;  
}

The column-reverse value stacks the flex items vertically (but from bottom to top):

.flex-container {  
  display: flex;  
  flex-direction: column-reverse;  
}

The row value stacks the flex items horizontally (from left to right):

.flex-container {  
  display: flex;  
  flex-direction: row;  
}

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.

The wrap value specifies that the flex items will wrap if necessary:

.flex-container {  
  display: flex;  
  flex-wrap: wrap;  
}

The nowrap value specifies that the flex items will not wrap (this is default):

.flex-container {  
  display: flex;  
  flex-wrap: nowrap;  
}

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:

The center value aligns the flex items at the center of the container:

.flex-container {  
  display: flex;  
  justify-content: center;  
}

The flex-end value aligns the flex items at the end of the container:

.flex-container {  
  display: flex;  
  justify-content: flex-end;  
}

The space-around value displays the flex items with space before, between, and after the lines:

.flex-container {  
  display: flex;  
  justify-content: space-around;  
}

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.

The center value aligns the flex items in the middle of the container:

.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: center;  
}

The flex-start value aligns the flex items at the top of the container:

.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: flex-start;  
}

The flex-end value aligns the flex items at the bottom of the container:

.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: flex-end;  
}

The stretch value stretches the flex items to fill the container (this is default):

.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: stretch;  
}

The baseline value aligns the flex items such as their baselines aligns:

.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: baseline;  
}

## The align-content Property

The align-content property is used to align the flex lines.

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;  
}

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;  
}

The center value displays the flex lines in the middle of the container:

.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: center;  
}

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;  
}

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;  
}

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:

.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 flex item properties are:

* [order](https://www.w3schools.com/css/css3_flexbox_items.asp#order)
* [flex-grow](https://www.w3schools.com/css/css3_flexbox_items.asp#flex-grow)
* [flex-shrink](https://www.w3schools.com/css/css3_flexbox_items.asp#flex-shrink)
* [flex-basis](https://www.w3schools.com/css/css3_flexbox_items.asp#flex-basis)
* [flex](https://www.w3schools.com/css/css3_flexbox_items.asp#flex)
* [align-self](https://www.w3schools.com/css/css3_flexbox_items.asp#align-self)

## The order Property

The order property specifies the order of the flex items.

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.

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.

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.

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.

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.

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>

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 Flexbox

For example, if you want to create a two-column layout for most screen sizes, and a one-column layout for small screen sizes (such as phones and tablets), you can change the flex-direction from row to column at a specific breakpoint (800px in the example below):

.flex-container {  
  display: flex;  
  flex-direction: row;  
}  
  
/\* Responsive layout - makes a one column layout instead of a two-column layout \*/  
@media (max-width: 800px) {  
  .flex-container {  
    flex-direction: column;  
  }  
}

Another way is to change the percentage of the flex property of the flex items to create different layouts for different screen sizes. Note that we also have to include flex-wrap: wrap; on the flex container for this example to work:

.flex-container {  
  display: flex;  
  flex-wrap: wrap;  
}  
  
.flex-item-left {  
  flex: 50%;  
}  
  
.flex-item-right {  
  flex: 50%;  
}  
  
/\* Responsive layout - makes a one column layout instead of a two-column layout \*/  
@media (max-width: 800px) {  
  .flex-item-right, .flex-item-left {  
    flex: 100%;  
  }  
}

## 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.

## Grid Elements

A grid layout consists of a parent element, with one or more child elements.

<style>

.grid-container {

display: grid;

grid-template-columns: auto auto auto;

background-color: #2196F3;

padding: 10px;

}

.grid-item {

background-color: rgba(255, 255, 255, 0.8);

border: 1px solid rgba(0, 0, 0, 0.8);

padding: 20px;

font-size: 30px;

text-align: center;

}

</style>

</head>

<body>

<h1>Grid Elements</h1>

<p>A Grid Layout must have a parent element with the <em>display</em> property set to <em>grid</em> or <em>inline-grid</em>.</p>

<p>Direct child element(s) of the grid container automatically becomes grid items.</p>

<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>

## Display Property

An HTML element becomes a grid container when its display property is set to grid or inline-grid.

.grid-container {  
  display: grid;  
}

.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:

* column-gap
* row-gap
* gap

The column-gap property sets the gap between the columns:

.grid-container {  
  display: grid;  
**column-gap: 50px;**  
}

The row-gap property sets the gap between the rows:

.grid-container {  
  display: grid;  
**row-gap: 50px;**}

The gap property is a shorthand property for the row-gap and the column-gap properties:

.grid-container {  
  display: grid;  
**gap: 50px 100px;**}

The gap property can also be used to set both the row gap and the column gap in one value:

.grid-container {  
  display: 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:

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;  
}

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;  
}

## 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 width 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.

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.

et 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.

.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.

.grid-container {  
  display: grid;  
  justify-content: space-evenly;  
}

.grid-container {  
  display: grid;  
  justify-content: space-around;  
}

.grid-container {  
  display: grid;  
  justify-content: space-between;  
}

.grid-container {  
  display: grid;  
  justify-content: center;  
}

.grid-container {  
  display: grid;  
  justify-content: start;  
}

.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.

.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: center;  
}

.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: space-evenly;  
}

.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: space-around;  
}

.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: space-between;  
}

.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: start;  
}

.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: end;  
}

## 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.

Make "item1" start on column 1 and end before column 5:

.item1 {  
  grid-column: 1 / 5;  
}

Make "item1" start on column 1 and span 3 columns:

.item1 {  
  grid-column: 1 / span 3;  
}

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:

Make "item1" start on row-line 1 and end on row-line 4:

.item1 {  
  grid-row: 1 / 4;  
}

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.

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;  
}

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.

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.

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:

Make "item1" span two columns *and* two rows:

.grid-container {  
  grid-template-areas: 'myArea myArea . . .' 'myArea myArea . . .';  
}

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.

.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:

@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; }  
}

**3. Containers**

Bootstrap also requires a containing element to wrap site contents.

There are two container classes to choose from:

1. The .container class provides a responsive **fixed width container**
2. The .container-fluid class provides a **full width container**, spanning the entire width of the viewport