

CSS-CASCADING STYLE SHEETS

CSS is the language we use to style a Web page.

What is CSS ?

CSS stands for Cascading Style Sheets

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

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

External stylesheets are stored in CSS files

Why Use CSS ?

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

eg:

```
body {  
    background-color: lightblue;  
}  
  
h1 {  
    color: white;  
    text-align: center;  
}  
  
p {  
    font-family: verdana;  
    font-size: 20px;  
}
```

CSS Solved a Big Problem:

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

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

<h1>This is a heading</h1>

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

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

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

CSS removed the style formatting from the HTML page!

CSS Saves a Lot of Work!:

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

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

CSS Syntax

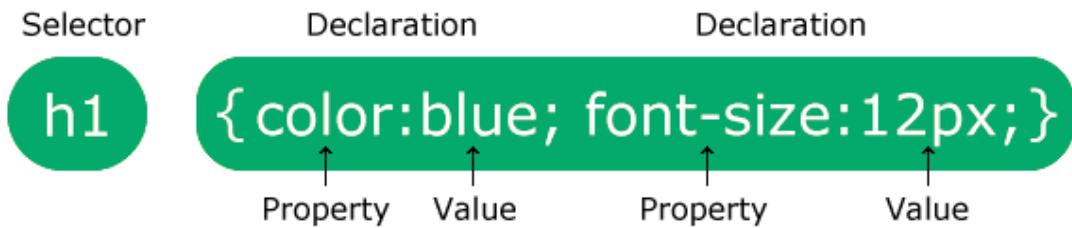
A CSS rule consists of a selector and a declaration block.

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

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

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

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



eg:

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

Example Explained

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

color is a property, and red is the property value

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

CSS Selectors

A CSS selector selects the HTML element(s) you want to style.

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

We can divide CSS selectors into five categories:

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

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

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

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

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

The CSS element Selector

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

eg:

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

The CSS id Selector

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

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

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

eg:

```
#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.

eg:

```
.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;
}
# 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;
}
```

The CSS Grouping Selector

The grouping selector selects all the HTML elements with the same style definitions.
Look at the following CSS code (the h1, h2, and p elements have the same style definitions):

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

```

color: red;
}

```

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

To group selectors, separate each selector with a comma.

eg:

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

```

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

```

All CSS Simple Selectors

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

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

External CSS

Internal CSS

Inline CSS

- External CSS:

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

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

eg:

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

```

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

```

```
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

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

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

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

```
"mystyle.css"
body {
    background-color: lightblue;
}
h1 {
    color: navy;
    margin-left: 20px;
}
```

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

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

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

- Internal CSS:

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

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

eg:

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

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

- Inline CSS:

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

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

eg:

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

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

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

- Multiple Style Sheets:

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

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

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

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

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

eg:

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

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

eg:

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

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

- Cascading Order:

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

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

Inline style (inside an HTML element)

External and internal style sheets (in the head section)

Browser default

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

CSS Comments

CSS comments are not displayed in the browser, but they can help document your source code.

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

Comments are ignored by browsers.

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

eg:

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

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

You can add comments wherever you want in the code:

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

Even in the middle of a code line:

```
p {  
    color: /*red*/blue;  
}
```

Comments can also span multiple lines:

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

- HTML and CSS Comments:

you can add comments to your HTML source by using the <!--...--> syntax.

In the following example, we use a combination of HTML and CSS comments:

eg:

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
p {  
    color: red; /* Set text color to red */  
}  
</style>  
</head>
```

```

<body>
<h2>My Heading</h2>
<!-- These paragraphs will be red -->
<p>Hello World!</p>
<p>This paragraph is styled with CSS.</p>
<p>CSS comments are not shown in the output.</p>
</body>
</html>

```

CSS Colors

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

HTML/CSS Supports 140 standard color names, to know please visit this link:

https://www.w3schools.com/colors/colors_names.asp

- CSS Background Color:

You can set the background color for HTML elements:

eg:

```

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

```

- CSS Text Color:

You can set the color of text:

eg:

```

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

```

- CSS Border Color:

You can set the color of borders:

```

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

```

- CSS Color Values:

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

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

eg:

```

h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>
<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>

```

CSS Backgrounds

The CSS background properties are used to add background effects for elements.

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

- `background-color`
- `background-image`
- `background-repeat`
- `background-attachment`
- `background-position`
- `background` (shorthand property)

• CSS background-color:

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

eg:

The background color of a page is set like this:

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

Other Elements:

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

```
h1 {  
    background-color: green;  
}  
  
div {  
    background-color: lightblue;  
}  
  
p {  
    background-color: yellow;  
}
```

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:



eg:

```
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. The following example sets the opacity for the background color and not the text:



you can use RGB as a color value. In addition to RGB, you can use an RGB color value with an alpha channel (RGBA) - which specifies the opacity for a color.

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

eg:

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

- CSS background-image:

The `background-image` property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element.

Set the background image for a page:

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

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:

eg:

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

- CSS background-repeat:

By default, the `background-image` property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange.

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

eg:

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

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

CSS background-repeat: no-repeat:

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

Show the background image only once:

eg:

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

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

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

CSS background-position:

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

Position the background image in the top-right corner:

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

CSS background-attachment:

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

Specify that the background image should be fixed:

eg:

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

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

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

CSS Background Shorthand

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

It does not matter if one of the property values is missing, as long as the other ones are in this order. # Note that we do not use the background-attachment property in the examples above, as it does not have a value

All CSS Background Properties

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

CSS Borders

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

- CSS Border Style:

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

The following values are allowed:

dotted - Defines a dotted border

dashed - Defines a dashed border

solid - Defines a solid border

double - Defines a double border

groove - Defines a 3D grooved border. The effect depends on the border-color value

ridge - Defines a 3D ridged border. The effect depends on the border-color value

inset - Defines a 3D inset border. The effect depends on the border-color value

outset - Defines a 3D outset border. The effect depends on the border-color value

none - Defines no border

hidden - Defines a hidden border

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

eg:

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

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

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

```

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

# Note: None of the OTHER CSS border properties (which you will learn more about in the
next chapters) will have ANY effect unless the border-style property is set!

```

- CSS Border Width:

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

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

eg:

```

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

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

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

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

```

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):

```

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

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

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

```

- CSS Border color:

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

The color can be set by:

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

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

eg:

```
p.one {
    border-style: solid;
    border-color: red;
}
p.two {
    border-style: solid;
    border-color: green;
}
p.three {
    border-style: dotted;
    border-color: blue;
}
```

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).

eg:

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

CSS Border Sides:

CSS Border - Individual Sides

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

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

eg:

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

So, here is how it works:

If the border-style property has four values:

border-style: dotted solid double dashed;

top border is dotted

right border is solid

bottom border is double

left border is dashed

If the border-style property has three values:

border-style: dotted solid double;

top border is dotted

right and left borders are solid

bottom border is double

If the border-style property has two values:

border-style: dotted solid;

top and bottom borders are dotted

right and left borders are solid

If the border-style property has one value:

border-style: dotted;

all four borders are dotted

- CSS Shorthand Border Property:

There are many properties to consider when dealing with borders.

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

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

border-width

border-style (required)

border-color

eg:

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

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

Left Border

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

Bottom Border

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

- CSS Rounded Borders:

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

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

<u>border</u>	Sets all the border properties in one declaration
<u>border-bottom</u>	Sets all the bottom border properties in one declaration
<u>border-bottom-color</u>	Sets the color of the bottom border
<u>border-bottom-style</u>	Sets the style of the bottom border
<u>border-bottom-width</u>	Sets the width of the bottom border
<u>border-color</u>	Sets the color of the four borders
<u>border-left</u>	Sets all the left border properties in one declaration
<u>border-left-color</u>	Sets the color of the left border
<u>border-left-style</u>	Sets the style of the left border
<u>border-left-width</u>	Sets the width of the left border
<u>border-radius</u>	Sets all the four border-* -radius properties for rounded corners
<u>border-right</u>	Sets all the right border properties in one declaration
<u>border-right-color</u>	Sets the color of the right border
<u>border-right-style</u>	Sets the style of the right border
<u>border-right-width</u>	Sets the width of the right border
<u>border-style</u>	Sets the style of the four borders
<u>border-top</u>	Sets all the top border properties in one declaration
<u>border-top-color</u>	Sets the color of the top border
<u>border-top-style</u>	Sets the style of the top border
<u>border-top-width</u>	Sets the width of the top border
<u>border-width</u>	Sets the width of the four borders

CSS Margins

Margins 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.

eg:

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

eg:

```

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

```

If the margin property has three values:

margin: 25px 50px 75px;

top margin is 25px

right and left margins are 50px

bottom margin is 75px

eg:

```

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

```

If the margin property has two values:

margin: 25px 50px;

top and bottom margins are 25px

right and left margins are 50px

eg:

```

p {
  margin: 25px 50px;
}

```

If the margin property has one value:

margin: 25px;

all four margins are 25px

eg:

```

p {
  margin: 25px;
}

```

- The Auto value:

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

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

eg:

```
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>`).

eg:

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

- Margin Collapse:

Sometimes two margins collapse into a single margin.

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:

Demonstration of margin collapse:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
    margin: 0 0 50px 0;
}
h2 {
    margin: 20px 0 0 0;
}
</style>
</head>
<body>
```

`<p>`In this example the h1 element has a bottom margin of 50px and the h2 element has a top margin of 20px. So, the vertical margin between h1 and h2 should have been 70px (50px + 20px). However, due to margin collapse, the actual margin ends up being 50px.`</p>`

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
</body>
```

</html>

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

Padding is used to create 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.

eg:

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

eg:

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

}

If the padding property has three values:

padding: 25px 50px 75px;

top padding is 25px

right and left paddings are 50px

bottom padding is 75px

eg:

div {

 padding: 25px 50px 75px;

}

If the padding property has two values:

padding: 25px 50px;

top and bottom paddings are 25px

right and left paddings are 50px

eg:

div {

 padding: 25px 50px;

}

If the padding property has one value:

padding: 25px;

all four paddings are 25px

eg:

div {

 padding: 25px;

}

• Padding and Element width:

The CSS width property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element.

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):

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div.ex1 {
  width: 300px;
  background-color: yellow;
}
div.ex2 {
  width: 300px;
  padding: 25px;
  background-color: lightblue;
}
</style>
</head>
```

```

<body>
<h2>Padding and element width</h2>
<div class="ex1">This div is 300px wide.</div>
<br>
<div class="ex2">The width of this div is 350px, even though it is defined as 300px in the
CSS.</div>
</body>
</html>

```

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.

eg:

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 Height, Width & Max-Width

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

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

- Setting height and Width:

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

eg:

Set the height and width of a <div> element:

```

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

```

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

- Setting Max-Width:

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

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

The problem with the <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).

eg:

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

All CSS Dimensions Properties:

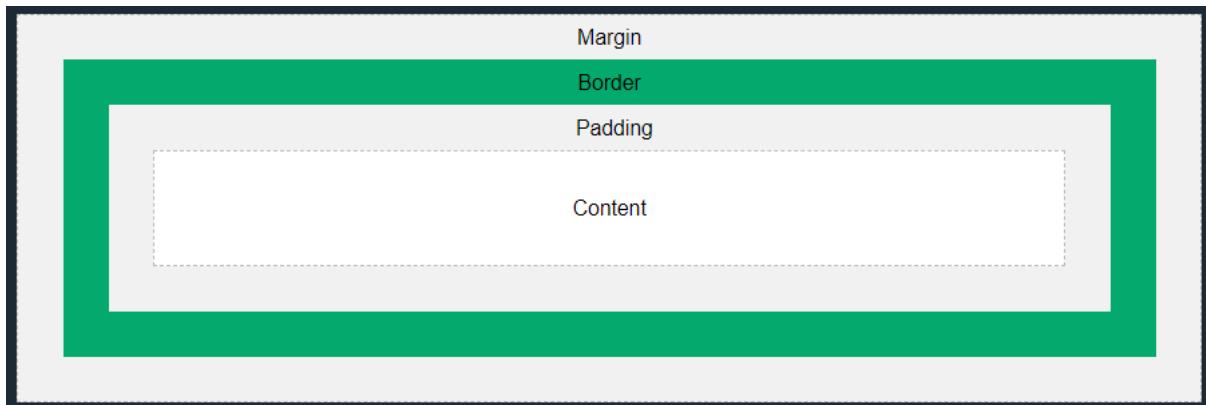
Property	Description
height	Sets the height of an element
max-height	Sets the maximum height of an element
max-width	Sets the maximum width of an element
min-height	Sets the minimum height of an element
min-width	Sets the minimum width of an element
width	Sets the width of an element

CSS Box Model

All HTML elements can be considered as boxes.

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

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



Explanation of the different parts:

Content - The content of the box, where text and images appear

Padding - Clears an area around the content. The padding is transparent

Border - A border that goes around the padding and content

Margin - Clears an area outside the border. The margin is transparent

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

eg:

Demonstration of the box model:

```
div {
    width: 300px;
    border: 15px solid green;
```

```

padding: 50px;
margin: 20px;
}

```

- Width and Height of an Element:

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

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

eg:

exact code: https://www.w3schools.com/css/tryit.asp?filename=trycss_boxmodel_width

```

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.

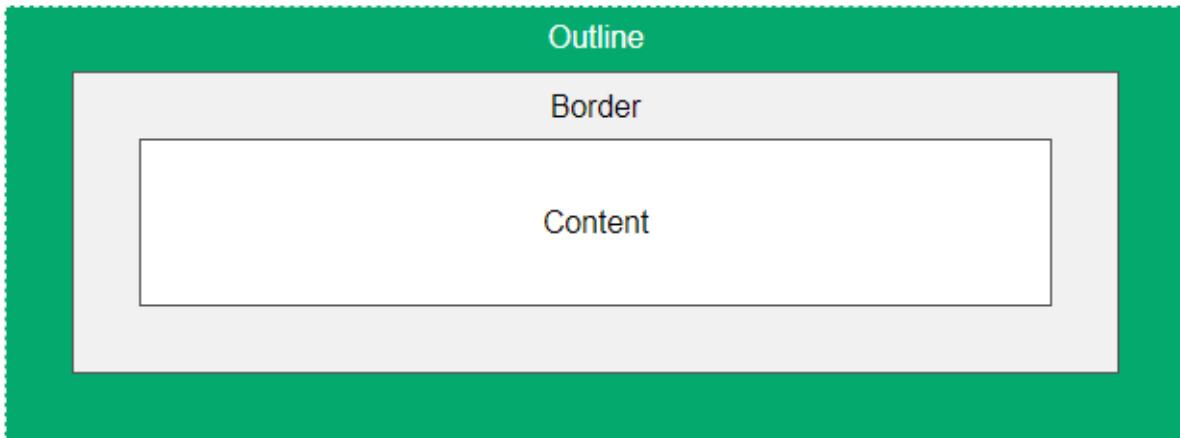
CSS Outline

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

eg:

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

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



CSS has the following outline properties:

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

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

- CSS Outline Style:

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

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

The following example shows the different `outline-style` values

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

p.dashed {outline-style: dashed;}  

p.solid {outline-style: solid;}  

p.double {outline-style: double;}  

p.groove {outline-style: groove;}  

p.ridge {outline-style: ridge;}  

p.inset {outline-style: inset;}
```

`p.outset {outline-style: outset;}`utline differs from borders! Unlike border, the outline is drawn outside the element's border, and may overlap other content. Also, the outline is NOT a part of the element's dimensions; the element's total width and height is not affected by the width of the outline.

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

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

The following example shows the different outline-style values

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

• 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)

eg:

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

```

border: 1px solid black;
outline-style: solid;
outline-color: red;
outline-width: 4px;
}

```

- CSS Outline Color:

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

The color can be set by:

name - specify a color name, like "red"

HEX - specify a hex value, like "#ff0000"

RGB - specify a RGB value, like "rgb(255,0,0)"

HSL - specify a HSL value, like "hsl(0, 100%, 50%)"

invert - performs a color inversion (which ensures that the outline is visible, regardless of color background)

- CSS Outline Shorthand:

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.

eg:

```

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.

eg:

```

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

```

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

CSS Text

CSS has a lot of properties for formatting text.

- Text color:

The color property is used to set the color of the text.

```

body {
  color: blue;
}
h1 {
  color: green;
}

```

~Text color & Background color:

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

```

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

```

- Text Alignment:

~ Text Alignment:

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

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

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

eg:

```

h1 {
  text-align: center;
}
h2 {
  text-align: left;
}
h3 {
  text-align: right;
}
```

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

eg:

```

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

Text Align Last:

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:

eg:

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

- Vertical Alignment:

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

eg:

```
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;
}
```

- CSS 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.

eg:

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

```
text-decoration-line: overline underline;
}
```

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

~ Specify a Color for the Decoration Line:

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

eg:

```
h1 {
    text-decoration-line: overline;
    text-decoration-color: red;
}
```

```
h2 {
    text-decoration-line: line-through;
    text-decoration-color: blue;
}
```

```
h3 {
    text-decoration-line: underline;
    text-decoration-color: green;
}
```

```
p {
    text-decoration-line: overline underline;
    text-decoration-color: purple;
}
```

~ Specify a Style for the Decoration Line:

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

eg:

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

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

```
h3 {
    text-decoration-line: underline;
    text-decoration-style: dotted;
}
```

```
p.ex1 {
    text-decoration-line: underline;
    text-decoration-style: dashed;
}
```

```
p.ex2 {
    text-decoration-line: underline;
    text-decoration-style: wavy;
}
```

```
p.ex3 {
    text-decoration-line: underline;
```

```

text-decoration-color: red;
text-decoration-style: wavy;
}

```

~ Specify the Thickness for the Decoration Line:

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

```

h1 {
    text-decoration-line: underline;
    text-decoration-thickness: auto;
}
h2 {
    text-decoration-line: underline;
    text-decoration-thickness: 5px;
}
h3 {
    text-decoration-line: underline;
    text-decoration-thickness: 25%;
}
p {
    text-decoration-line: underline;
    text-decoration-color: red;
    text-decoration-style: double;
    text-decoration-thickness: 5px;
}

```

~ The Shorthand property:

The text-decoration property is a shorthand property for:

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

eg:

```

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

```

A Small Tip:

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:

eg:

```

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

}

- CSS Text Transformaion:

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;
}
```

- CSS Text Spacing:

- ~ Text Indentation:

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

eg:

```
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:

eg:

```
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:

eg:

```
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:

eg:

```
p {
  white-space: nowrap;
}
```

• CSS 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)

eg 1:

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

eg 2:

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

Some of the best Practices:

1) Text-shadow on a white text:

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

2) Text-shadow with red neon glow:

```
h1 {
  text-shadow: 0 0 3px #ff0000;
}
```

3) Text-shadow with red and blue neon glow:

```
h1 {
  text-shadow: 0 0 3px #ff0000, 0 0 5px #0000ff;
}
```

4) h1 {
 color: white;
 text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px darkblue;
}

CSS Fonts

Choosing the right font for your website is important!

Choosing the right font has a huge impact on how the readers experience a website.

The right font can create a strong identity for your brand.

Using a font that is easy to read is important. The font adds value to your text. It is also important to choose the correct color and text size for the font.

~ Generic Font Families:

In CSS there are five generic font families:

Serif fonts have a small stroke at the edges of each letter. They create a sense of formality and elegance.

Sans-serif fonts have clean lines (no small strokes attached). They create a modern and minimalistic look.

Monospace fonts - here all the letters have the same fixed width. They create a mechanical look.

Cursive fonts imitate human handwriting.

Fantasy fonts are decorative/playful fonts.

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

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

- Font-family Property:

eg:

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

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

- 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.

eg:

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)

• Font Fallbacks:

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

Serif
 Sans-serif
 Monospace
 Cursive
 Fantasy

For more visit: https://www.w3schools.com/css/css_font_fallbacks.asp

• 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)

eg:

```
p.normal {  

    font-style: normal;  

}  

p.italic {  

    font-style: italic;  

}  

p.oblique {  

    font-style: oblique;  

}
```

~ Font Weight:

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

```
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.

eg:

```

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

```

• Font Size:

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

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

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

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

Absolute size:

Sets the text to a specified size

Does not allow a user to change the text size in all browsers (bad for accessibility reasons)

Absolute size is useful when the physical size of the output is known

Relative size:

Sets the size relative to surrounding elements

Allows a user to change the text size in browsers

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

~ Set Font Size With Pixels:

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

```

h1 {
  font-size: 40px;
}
h2 {
  font-size: 30px;
}
p {
  font-size: 14px;
}
```

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

~ Set Font Size With Em:

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

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

eg:

```

h1 {
  font-size: 2.5em; /* 40px/16=2.5em */
}
h2 {
  font-size: 1.875em; /* 30px/16=1.875em */
}
p {
```

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

~ Responsive Font Size:

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

eg:

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

Note: 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.

eg:

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

Note: When specifying a font in CSS, always list at minimum one fallback font (to avoid unexpected behaviors). So, also here you should add a generic font family (like serif or sans-serif) to the end of the list.

To know list os all Google Fonts: https://www.w3schools.com/howto/howto_google_fonts.asp

~ Some of the best practices:

- Styling Google Fonts:

Of course you can style Google Fonts as you like, with CSS!

eg:

Style the "Sofia" font:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">
<style>
body {
  font-family: "Sofia", sans-serif;
  font-size: 30px;
  text-shadow: 3px 3px 3px #ababab;
}
</style>
</head>
```

- Enabling Font Effects:

Google has also enabled different font effects that you can use.

First add effect=effectname to the Google API, then add a special class name to the element that is going to use the special effect. The class name always starts with font-effect- and ends with the effectname.

eg:

Add the fire effect to the "Sofia" font:

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia&effect=fire">
<style>
body {
  font-family: "Sofia", sans-serif;
  font-size: 30px;
}
</style>
</head>
<body>
<h1 class="font-effect-fire">Sofia on Fire</h1>
</body>
```

~~ 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-m
      ultiple">
<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>
```

- Great Font Pairings:

Great font pairings are essential to great design.

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.

eg:

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

~ Some Great font pairings:

1) Georgia and Verdana

Georgia and Verdana is a classic combination. It also sticks to the web safe font standards

eg:

```
h1 {  
  font-family: Georgia, serif;  
  font-size: 46px;  
}
```

2) Helvetica and Garamond

Helvetica and Garamond is another classic combination that uses web safe fonts.

eg:

```
h1 {  
  font-family: Helvetica, sans-serif;  
  font-size: 46px;  
}
```

~ ~ Google font pairings:

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

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

Below are some popular Google Web Font Pairings.

1) Merriweather and Open Sans

```
h1 {  
  font-family: Merriweather, serif;  
  font-size: 46px;  
}
```

2) Ubuntu and Lora

```
h1 {  
  font-family: Ubuntu, sans-serif;  
  font-size: 46px;  
}
```

3) Abril Fatface and Poppins

```

h1 {
  font-family: 'Abril Fatface', serif;
  font-size: 46px;
}
4) Cinzel and Fauna One
h1 {
  font-family: Cinzel, serif;
  font-size: 46px;
}
5) Fjalla One and Libre Baskerville
h1 {
  font-family: 'Fjalla One', sans-serif;
  font-size: 46px;
}
6) Space Mono and Muli
h1 {
  font-family: "Space Mono", monospace;
  font-size: 46px;
}
7) Spectral and Rubik
h1 {
  font-family: Spectral, serif;
  font-size: 46px;
}
8) Oswald and Noto Sans
h1 {
  font-family: Oswald, sans-serif;
  font-size: 46px;
}

```

• Font Shorthand:

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 values are used.

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

eg:

Use font to set several font properties in one declaration:

```

p.a {
  font: 20px Arial, sans-serif;
}
p.b { font: italic small-caps bold 12px/30px Georgia, serif;}
```

CSS Icons

How To Add Icons ?

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

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

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

- Font Awesome Icons:

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

```
<script src="https://kit.fontawesome.com/yourcode.js" crossorigin="anonymous"></script>
```

Note: No downloading or installation is required!

eg:

```
<!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">
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
</head>
<body>
<i class="glyphicon glyphicon-cloud"></i>
<i class="glyphicon glyphicon-remove"></i>
<i class="glyphicon glyphicon-user"></i>
<i class="glyphicon glyphicon-envelope"></i>
<i class="glyphicon glyphicon-thumbs-up"></i>
</body>
</html>
```

- 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">
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
</head>
<body>
<i class="material-icons">cloud</i>
<i class="material-icons">favourite</i>
<i class="material-icons">attachment</i>
<i class="material-icons">computer</i>
<i class="material-icons">traffic</i>
</body>
</html>
```

CSS Links

- 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

eg:

```
/* unvisited link */
```

```
a:link {
  color: red;
}
```

```
/* visited link */
```

```
a:visited {
  color: green;
}
```

```
/* mouse over link */
```

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

```
/* selected link */
```

```
a:active {
  color: blue;
```

```
}
```

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

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

a:active MUST come after a:hover

- Text Decoration:

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

eg:

```
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:

eg:

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

- Link Buttons:

```
<!DOCTYPE html>
<html>
<head>
<style>
a:link, a:visited {
    background-color: #f44336;
    color: white;
    padding: 14px 25px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
}
a:hover, a:active {
```

```

background-color: red;
}
</style>
</head>
<body>
<h2>Link Button</h2>
<p>A link styled as a button:</p>
<a href="default.asp" target="_blank">This is a link</a>
</body>
</html>
• More Examples:
This example demonstrates how to add other styles to hyperlinks:
<!DOCTYPE html>
<html>
<head>
<style>
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;}
</style>
</head>
<body>
<h2>Styling Links</h2>
<p>Mouse over the links and watch them change layout:</p>
<p><b><a class="one" href="default.asp" target="_blank">This link changes color</a></b></p>
<p><b><a class="two" href="default.asp" target="_blank">This link changes font-size</a></b></p>
<p><b><a class="three" href="default.asp" target="_blank">This link changes background-color</a></b></p>
<p><b><a class="four" href="default.asp" target="_blank">This link changes font-family</a></b></p>

```

```

<p><b><a class="five" href="default.asp" target="_blank">This link changes
text-decoration</a></b></p>
</body>
</html>
~ Another example of how to create link boxes/buttons:
a:link, a:visited {
    background-color: white;
    color: black;
    border: 2px solid green;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
}
a:hover, a:active {
    background-color: green;
    color: white;
}
~ This example demonstrates the different types of cursors (can be useful for links):
<!DOCTYPE html>
<html>
<body>
<h2>The cursor Property</h2>
<p>Mouse over the words to change the cursor.</p>
<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><br>
</body>
</html>

```

CSS Lists

- HTML Lists and CSS List Properties:

In HTML, there are two main types of lists:

unordered lists (``) - the list items are marked with bullets

ordered lists (``) - the list items are marked with numbers or letters

The CSS list properties allow you to:

Set different list item markers for ordered lists

Set different list item markers for unordered lists

Set an image as the list item marker

Add background colors to lists and list items

- Different List Item Markers:

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

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

eg:

```
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 maker:

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

eg:

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

- Position the list item markers:

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

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

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

eg:

```
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 `` or ``:

eg:

```
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:

eg:

```
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 `` or `` tag, affects the entire list, while properties added to the `` tag will affect the individual list items:

eg:

```
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;
}
```

- More Examples:

~ List with a red left border:

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    border-left: 5px solid red;
    background-color: #f1f1f1;
    list-style-type: none;
    padding: 10px 20px;
}
</style>
</head>
<body>
<h2>List with a red left border</h2>
<ul>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Coca Cola</li>
</ul>
</body>
</html>
```

~ A bordered list:

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    list-style-type: none;
    padding: 0;
    border: 1px solid #ddd;
}
ul li {
    padding: 8px 16px;
    border-bottom: 1px solid #ddd;
}
ul li:last-child {
    border-bottom: none
}
</style>
</head>
<body>
<h2>A bordered list</h2>
<ul>
    <li>Coffee</li>
    <li>Tea</li>
    <li>Coca Cola</li>
```

```

</ul>
</body>
</html>
~ All list types:
<!DOCTYPE html>
<html>
<head>
<style>
ul.a {list-style-type: circle;}
ul.b {list-style-type: disc;}
ul.c {list-style-type: square;}
ol.d {list-style-type: armenian;}
ol.e {list-style-type: cjk-ideographic;}
ol.f {list-style-type: decimal;}
ol.g {list-style-type: decimal-leading-zero;}
ol.h {list-style-type: georgian;}
ol.i {list-style-type: hebrew;}
ol.j {list-style-type: hiragana;}
ol.k {list-style-type: hiragana-iroha;}
ol.l {list-style-type: katakana;}
ol.m {list-style-type: katakana-iroha;}
ol.n {list-style-type: lower-alpha;}
ol.o {list-style-type: lower-greek;}
ol.p {list-style-type: lower-latin;}
ol.q {list-style-type: lower-roman;}
ol.r {list-style-type: upper-alpha;}
ol.s {list-style-type: upper-latin;}
ol.t {list-style-type: upper-roman;}
ol.u {list-style-type: none;}
ol.v {list-style-type: inherit;}
</style>
</head>
<body>
<h2>All List Style Types</h2>
<ul class="a">
<li>Circle type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ul>
<ul class="b">
<li>Disc type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ul>
<ul class="c">
<li>Square type</li>
<li>Tea</li>
<li>Coca Cola</li>

```

```
</ul>
<ol class="d">
  <li>Armenian type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="e">
  <li>Cjk-ideographic type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="f">
  <li>Decimal type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="g">
  <li>Decimal-leading-zero type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="h">
  <li>Georgian type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="i">
  <li>Hebrew type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="j">
  <li>Hiragana type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="k">
  <li>Hiragana-iroha type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="l">
  <li>Katakana type</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ol>
<ol class="m">
  <li>Katakana-iroha type</li>
```

```
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="n">
<li>Lower-alpha type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="o">
<li>Lower-greek type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="p">
<li>Lower-latin type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="q">
<li>Lower-roman type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="r">
<li>Upper-alpha type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="s">
<li>Upper-latin type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="t">
<li>Upper-roman type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="u">
<li>None type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol>
<ol class="v">
<li>inherit type</li>
<li>Tea</li>
<li>Coca Cola</li>
</ol></body></html>
```

CSS Tables

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

eg:

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

- Full-Width Table:

If you need a table that should span the entire screen (full-width), add width: 100% to the <table> element.

eg:

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

Note: the table above will give double borders. This is because both the table and the <th> and <td> elements have separate borders.

To remove double borders:

Collapse Table Borders:

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

eg:

```
table {  
    border-collapse: collapse;  
}
```

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

eg:

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

- Table Size:

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

eg:

```
table {  
    width: 100%;  
}  
  
th {  
    height: 70px;  
}
```

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

```
table {  
    width: 50%;  
}
```

- Table Alignment:

~ 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

eg:

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

To left-align the content, force the alignment of <th> elements to be left-aligned, with the text-align: left property:

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

~ Vertical Alignment:

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

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

eg:

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

• Table Style:

~ Table Padding:

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

eg:

```
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;
}
```

~ Hover Table:

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

eg:

```
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

eg:

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

~ Table Color:

```
th {
    background-color: #04AA6D;
    color: white;
}
```

• CSS 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.

eg:

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

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

- [More Examples](#):

~ A fancy Table:

```
<!DOCTYPE html>
<html>
<head>
<style>
#customers {
    font-family: Arial, Helvetica, sans-serif;
    border-collapse: collapse;
    width: 100%;
}
#customers td, #customers th {
    border: 1px solid #ddd;
    padding: 8px;
}
#customers tr:nth-child(even){background-color: #f2f2f2;}
#customers tr:hover {background-color: #ddd;}
#customers th {
    padding-top: 12px;
    padding-bottom: 12px;
    text-align: left;
    background-color: #04AA6D;
    color: white;
}
</style>
</head>
<body>
<h1>A Fancy Table</h1>
<table id="customers">
<tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
</tr>
<tr>
    <td>Alfreds Futterkiste</td>
```

```
<td>Maria Anders</td>
<td>German</td>
</tr>
<tr>
<td>Berglunds snabbköp</td>
<td>Christina Berglund</td>
<td>Sweden</td>
</tr>
<tr>
<td>Centro comercial Moctezuma</td>
<td>Francisco Chang</td>
<td>Mexico</td>
</tr>
<tr>
<td>Ernst Handel</td>
<td>Roland Mendel</td>
<td>Austria</td>
</tr>
<tr>
<td>Island Trading</td>
<td>Helen Bennett</td>
<td>UK</td>
</tr>
<tr>
<td>Königlich Essen</td>
<td>Philip Cramer</td>
<td>German</td>
</tr>
<tr>
<td>Laughing Bacchus Wine Cellars</td>
<td>Yoshi Tannamuri</td>
<td>Canada</td>
</tr>
<tr>
<td>Magazzini Alimentari Riuniti</td>
<td>Giovanni Rovelli</td>
<td>Italy</td>
</tr>
<tr>
<td>North/South</td>
<td>Simon Crowther</td>
<td>UK</td>
</tr>
<tr>
<td>Paris spécialités</td>
<td>Marie Bertrand</td>
<td>France</td>
</tr>
```

```

</table>
</body>
</html>
~ Caption position:
caption {
  caption-side: bottom;
}

```

- CSS Table Properties:

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

CSS Layout - The display property

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

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

Every HTML element has a default display value depending on what type of element it is.

The default display value for most elements is block or inline.

~ Block Level Elements:

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

Examples of block-level elements:

```

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

```

~ Inline Level Elements:

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

This is an inline `` element inside a paragraph.

Examples of inline elements:

```

<span>
<a>
<img>

```

- Display None:

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

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 `` elements for horizontal menus

eg:

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

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

~ The following example displays `` elements as block elements:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
span {
    display: block;
}
</style>
</head>
<body>
<h1>Display span elements as block elements</h1>
<span>A display property with</span> <span>a value of "block" results in</span> <span>a line break between each span elements.</span>
</body>
</html>
```

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
a {
    display: block;
}
</style>
</head>
<body>
<h1>Display links as block elements</h1>
<a href="/html/default.asp" target="_blank">HTML</a>
<a href="/css/default.asp" target="_blank">CSS</a>
<a href="/js/default.asp" target="_blank">JavaScript</a>
</body>
</html>
```

- 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:

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

~ 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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1.hidden {
  visibility: hidden;
}
</style>
</head>
<body>
<h1>This is a visible heading</h1>
<h1 class="hidden">This is a hidden heading</h1>
<p>Notice that the hidden heading still takes up space.</p>
</body>
</html>
```

~ This example demonstrates display: none; versus visibility: hidden; :

```
<!DOCTYPE html>
<html>
<head>
<style>
.imgbox {
  float: left;
  text-align: center;
  width: 120px;
  border: 1px solid gray;
```

```

margin: 4px;
padding: 6px;
}
button {
width: 100%;
}
</style>
</head>
<body>
<h3>Difference between display:none and visibility: hidden</h3>
<p><strong>visibility:hidden</strong> hides the element, but it still takes up space in the layout.</p>
<p><strong>display:none</strong> removes the element from the document. It does not take up any space.</p>
<div class="imgbox" id="imgbox1">Box 1<br>

<button onclick="removeElement()">Remove</button>
</div>
<div class="imgbox" id="imgbox2">Box 2<br>

<button onclick="changeVisibility()">Hide</button>
</div>
<div class="imgbox">Box 3<br>

<button onclick="resetElement()">Reset All</button>
</div>
<script>
function removeElement() {
document.getElementById("imgbox1").style.display = "none";
}
function changeVisibility() {
document.getElementById("imgbox2").style.visibility = "hidden";
}
function resetElement() {
document.getElementById("imgbox1").style.display = "block";
document.getElementById("imgbox2").style.visibility = "visible";
}
</script>
</body>
</html>
~ This example demonstrates how to use CSS and JavaScript to show an element on click:
<!DOCTYPE html>
<html>
<head>
<style>
#panel, .flip {
font-size: 16px;
padding: 10px;

```

```

text-align: center;
background-color: #4CAF50;
color: white;
border: solid 1px #a6d8a8;
margin: auto;
}
#panel {
display: none;
}
</style>
</head>
<body>
<p class="flip" onclick="myFunction()">Click to show panel</p>
<div id="panel">
<p>This panel contains a div element, which is hidden by default (display: none).</p>
<p>It is styled with CSS and we use JavaScript to show it (display: block).</p>
<p>How it works: Notice that the p element with class="flip" has an onclick attribute attached to it. When the user clicks on the p element, a function called myFunction() is executed, which changes the style of the div with id="panel" from display:none (hidden) to display:block (visible).</p>
<p>You will learn more about JavaScript in our JavaScript Tutorial.</p>
</div>
<script>
function myFunction() {
document.getElementById("panel").style.display = "block";
}
</script>
</body>
</html>

```

CSS Layout - width and max-width

Using width, max-width and margin: auto;

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

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

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:

eg:

```
<!DOCTYPE html>
<html>
<head>
```

```

<style>
div.ex1 {
    width: 500px;
    margin: auto;
    border: 3px solid #73AD21;
}
div.ex2 {
    max-width: 500px;
    margin: auto;
    border: 3px solid #73AD21;
}
</style>
</head>
<body>
<h2>CSS Max-width</h2>
<div class="ex1">This div element has width: 500px;</div>
<br>
<div class="ex2">This div element has max-width: 500px;</div>
<p><strong>Tip:</strong> Drag the browser window to smaller than 500px wide, to see the
difference between
the two divs!</p>
</body>
</html>

```

CSS Layout - The position Property

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

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

There are five different position values:

- static
- relative
- fixed
- absolute
- sticky

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

- Position: static;

HTML elements are positioned static by default.

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

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>

```

```

div.static {
  position: static;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
<h2>position: static;</h2>
<p>An element with position: static; is not positioned in any special way; it is always
positioned according to the normal flow of the page:</p>
<div class="static">
This div element has position: static;
</div>
</body>
</html>
• 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.

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
<h2>position: relative;</h2>
<p>An element with position: relative; is positioned relative to its normal position:</p>
<div class="relative">
This div element has position: relative;
</div>
</body>
</html>
• 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:
eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
<h2>position: fixed;</h2>
<p>An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled:</p>
<div class="fixed">
This div element has position: fixed;
</div>
</body>
</html>

```

- **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.

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}
div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}

```

```

</style>
</head>
<body>
<h2>position: absolute;</h2>
<p>An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):</p>
<div class="relative">This div element has position: relative;
  <div class="absolute">This div element has position: absolute;</div>
</div>
</body>
</html>

```

- 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).

eg:

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

```

<!DOCTYPE html>
<html>
<head>
<style>
div.sticky {
  position: -webkit-sticky;
  position: sticky;
  top: 0;
  padding: 5px;
  background-color: #cae8ca;
  border: 2px solid #4CAF50;
}
</style>
</head>
<body>
<p>Try to <b>scroll</b> inside this frame to understand how sticky positioning works.</p>
<div class="sticky">I am sticky!</div>
<div style="padding-bottom:2000px">
  <p>In this example, the sticky element sticks to the top of the page (top: 0), when you reach its scroll position.</p>
  <p>Scroll back up to remove the stickyness.</p>
  <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>
  <p>Some text to enable scrolling.. Lorem ipsum dolor sit amet, illum definitiones no quo, maluisset concludaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.</p>

```

```

</div>
</body>
</html>
• Positioning Text in an Image:
~ Top Left:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.topleft {
  position: absolute;
  top: 8px;
  left: 16px;
  font-size: 18px;
}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the top left corner:</p>
<div class="container">
  
  <div class="topleft">Top Left</div>
</div>
</body>
</html>

~ Top Right:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.topright {
  position: absolute;
  top: 8px;
  right: 16px;
  font-size: 18px;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the top right corner:</p>
<div class="container">
  
  <div class="topright">Top Right</div>
</div>
</body>
</html>

```

```

}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the top right corner:</p>
<div class="container">
  
  <div class="topright">Top Right</div>
</div>
</body>
</html>

```

~ Bottom Left:

```

<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.bottomleft {
  position: absolute;
  bottom: 8px;
  left: 16px;
  font-size: 18px;
}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the bottom left corner:</p>
<div class="container">
  
  <div class="bottomleft">Bottom Left</div>
</div>
</body>
</html>

```

~ Bottom Right:

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.bottomright {
  position: absolute;
  bottom: 8px;
  right: 16px;
  font-size: 18px;
}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the bottom right corner:</p>
<div class="container">
  
  <div class="bottomright">Bottom Right</div>
</div>
</body>
</html>
```

~ Centered:

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.center {
  position: absolute;
  top: 50%;
  width: 100%;
  text-align: center;
  font-size: 18px;
}
img {
```

```

width: 100%;
height: auto;
opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Center text in image:</p>
<div class="container">
  
  <div class="center">Centered</div>
</div>
</body>
</html>
~ This example demonstrates how to set the shape of an element. The element is clipped
into this shape, and displayed.:
<!DOCTYPE html>
<html>
<head>
<style>
img {
  position: absolute;
  clip: rect(0px,60px,200px,0px);
}
</style>
</head>
<body>

</body>
</html>

```

CSS Layout - The z-index Property

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

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
img {
  position: absolute;
  left: 0px;
  top: 0px;

```

```

z-index: -1;
}
</style>
</head>
<body>
<h1>This is a heading</h1>

<p>Because the image has a z-index of -1, it will be placed behind the text.</p>
</body>
</html>
# Note: z-index only works on positioned elements (position: absolute, position: relative, position: fixed, or position: sticky) and flex items (elements that are direct children of display: flex elements).

```

- Another Z-Index example:

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

```

<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.black-box {
  position: relative;
  z-index: 1;
  border: 2px solid black;
  height: 100px;
  margin: 30px;
}
.gray-box {
  position: absolute;
  z-index: 3; /* gray box will be above both green and black box */
  background: lightgray;
  height: 60px;
  width: 70%;
  left: 50px;
  top: 50px;
}
.green-box {
  position: absolute;
  z-index: 2; /* green box will be above black box */
  background: lightgreen;
  width: 35%;
  left: 270px;
  top: -15px;
  height: 100px;
}

```

```

</style>
</head>
<body>
<h1>Z-index Example</h1>
<p>An element with greater stack order is always above an element with a lower stack
order.</p>
<div class="container">
  <div class="black-box">Black box (z-index: 1)</div>
  <div class="gray-box">Gray box (z-index: 3)</div>
  <div class="green-box">Green box (z-index: 2)</div>
</div>
</body>
</html>

```

- Without Z-Index:

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

```

<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.black-box {
  position: relative;
  border: 2px solid black;
  height: 100px;
  margin: 30px;
}
.gray-box {
  position: absolute;
  background: lightgray;
  height: 60px;
  width: 70%;
  left: 50px;
  top: 50px;
}
.green-box {
  position: absolute;
  background: lightgreen;
  width: 35%;
  left: 270px;
  top: -15px;
  height: 100px;
}
</style>
</head>
<body>

```

```

<h1>Overlapping elements</h1>
<p>If two positioned elements overlap each other without a z-index specified,
the element defined last in the HTML code will be shown on top:</p>
<div class="container">
  <div class="black-box">Black box</div>
  <div class="gray-box">Gray box</div>
  <div class="green-box">Green box</div>
</div>
</body>
</html>

```

CSS Layout - Overflow

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

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.

eg:

```

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.

eg:

```

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).

eg:

```

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.

eg:

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

CSS Layout - float and clear

The CSS float property specifies how an element should float.

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

Float Property:

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

The float property can have one of the following values:

left - The element floats to the left of its container

right - The element floats to the right of its container

none - The element does not float (will be displayed just where it occurs in the text). This is default

inherit - The element inherits the float value of its parent

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

eg1: *Float: right;*

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
  float: right;
}
</style>
</head>
<body>
<h2>Float Right</h2>
<p>In this example, the image will float to the right in the paragraph, and the text in the paragraph will wrap around the image.</p>
<p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>

```

</body>
</html>
eg2: Float: left;
<!DOCTYPE html>
<html>
<head>
<style>
img {
  float: left;
}
</style>
</head>
<body>
<h2>Float Left</h2>
<p>In this example, the image will float to the left in the paragraph, and the text in the paragraph will wrap around the image.</p>
<p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>
</body>
</html>
eg3: Float: none;
<!DOCTYPE html>
<html>
<head>
<style>
img {
  float: none;
}
</style>
</head>
<body>
<h2>Float None</h2>

```

```

<p>In this example, the image will be displayed just where it occurs in the text (float: none;).</p>
<p>
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum
interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas
nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut
aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a
lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum
dignissim ac. In at libero sed nunc venenatis imperdiet sed ornare turpis. Donec vitae dui
eget tellus gravida venenatis. Integer fringilla congue eros non fermentum. Sed dapibus
pulvinar nibh tempor porta. Cras ac leo purus. Mauris quis diam velit.</p>
</body>
</html>

```

eg4: Float Next To Each Other

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

```

<!DOCTYPE html>
<html>
<head>
<style>
div {
  float: left;
  padding: 15px;
}
.div1 {
  background: red;
}
.div2 {
  background: yellow;
}
.div3 {
  background: green;
}
</style>
</head>
<body>
<h2>Float Next To Each Other</h2>
<p>In this example, the three divs will float next to each other.</p>
<div class="div1">Div 1</div>
<div class="div2">Div 2</div>
<div class="div3">Div 3</div>
</body>
</html>

```

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.

eg:

This example clears the float to the left. Here, it means that the <div2> element is pushed below the left floated <div1> element.

```
<!DOCTYPE html>
<html>
<head>
<style>
.div1 {
  float: left;
  padding: 10px;
  border: 3px solid #73AD21;
}
.div2 {
  padding: 10px;
  border: 3px solid red;
}
.div3 {
  float: left;
  padding: 10px;
  border: 3px solid #73AD21;
}
.div4 {
  padding: 10px;
  border: 3px solid red;
  clear: left;
}
</style>
</head>
<body>
<h2>Without clear</h2>
<div class="div1">div1</div>
<div class="div2">div2 - Notice that div2 is after div1 in the HTML code. However, since div1
floats to the left, the text in div2 flows around div1.</div>
<br><br>
<h2>With clear</h2>
<div class="div3">div3</div>
<div class="div4">div4 - Here, clear: left; moves div4 down below the floating div3. The value
"left" clears elements floated to the left. You can also clear "right" and "both".</div>
</body>
```

</html>

- 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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    border: 3px solid #4CAF50;
    padding: 5px;
}
.img1 {
    float: right;
}
.img2 {
    float: right;
}
.clearfix {
    overflow: auto;
}
</style>
</head>
<body>
<h2>Without Clearfix</h2>
<p>This image is floated to the right. It is also taller than the element containing it, so it overflows outside of its container:</p>
<div>
    
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet...
</div>
<h2 style="clear:right">With Clearfix</h2>
<p>We can fix this by adding a clearfix class with overflow: auto; to the containing element:</p>
<div class="clearfix">
    
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet...
</div>
</body>
</html>
```

~ 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:

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```

div {
    border: 3px solid #4CAF50;
    padding: 5px;
}
.img1 {
    float: right;
}
.img2 {
    float: right;
}
.clearfix::after {
    content: "";
    clear: both;
    display: table;
}
</style>
</head>
<body>
<h2>Without Clearfix</h2>
<p>This image is floated to the right. It is also taller than the element containing it, so it overflows outside of its container:</p>
<div>
    
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet...
</div>
<h2 style="clear:right">With New Modern Clearfix</h2>
<p>Add the clearfix hack to the containing element, to fix this problem:</p>
<div class="clearfix">
    
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet...
</div>
</body>
</html>

```

Float Examples:

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

```

<!DOCTYPE html>
<html>
<head>
<style>
* {
    box-sizing: border-box;
}
.box {
    float: left;
    width: 33.33%;
    padding: 50px;
}
.clearfix::after {

```

```

content: "";
clear: both;
display: table;
}
</style>
</head>
<body>
<h2>Grid of Boxes</h2>
<p>Float boxes side by side:</p>
<div class="clearfix">
  <div class="box" style="background-color:#bbb">
    <p>Some text inside the box.</p>
  </div>
  <div class="box" style="background-color:#ccc">
    <p>Some text inside the box.</p>
  </div>
  <div class="box" style="background-color:#ddd">
    <p>Some text inside the box.</p>
  </div>
</div>
<p><strong>Note:</strong> Here, we use the clearfix hack to take care of the layout flow. We also use the box-sizing property to make sure that the box doesn't break due to extra padding. Try to remove this code to see the effect.</p>
</body>
</html>

```

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.

2) Images Side-by-Side:

```

<!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">
    
  </div>
  <div class="img-container">
    
  </div>
  <div class="img-container">
    
  </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>

```

3) 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.

```

<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}
.box {
  float: left;
  width: 50%;
  padding: 50px;
  height: 300px;
}
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
</style>
</head>
<body>
<h2>Equal Height Boxes</h2>
<p>Floating boxes with equal heights:</p>

```

```

<div class="clearfix">
  <div class="box" style="background-color:#bbb">
    <h2>Box 1</h2>
    <p>Some content, some content, some content</p>
  </div>
  <div class="box" style="background-color:#ccc">
    <h2>Box 2</h2>
    <p>Some content, some content, some content</p>
    <p>Some content, some content, some content</p>
    <p>Some content, some content, some content</p>
  </div>
</div>
<p>This example is not very flexible. It is ok to use CSS height if you can guarantee that the boxes will always have the same amount of content in them, but that's not always the case. If you try the example above on a mobile phone (or resize the browser window), you will see that the second box's content will be displayed outside of the box.</p>
<p>Go back to the tutorial and find another solution, if this is not what you want.</p>
</body>
</html>

```

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.

```

<!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: 75px;
  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>
4) Navigation Menu:
You can also use float with a list of hyperlinks to create a horizontal menu.
eg:
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
}
li {
    float: left;
}
li a {
    display: inline-block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}
li a:hover {
    background-color: #111;
}
.active {
    background-color: red;
}
</style>
</head>
<body>
<ul>
<li><a href="#home" class="active">Home</a></li>
<li><a href="#news">News</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
</body>
</html>

```

5) Web Layout:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}
body {
  background-color: white;
}
.header, .footer {
  background-color: grey;
  color: white;
  padding: 15px;
}
.column {
  float: left;
  padding: 15px;
}
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
.menu {
  width: 25%;
}
.content {
  width: 75%;
}
.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
.menu li {
  padding: 8px;
  margin-bottom: 8px;
  background-color: #33b5e5;
  color: #ffffff;
}
.menu li:hover {
  background-color: #0099cc;
}
</style>
```

```

</head>
<body>
<div class="header">
  <h1>Chania</h1>
</div>
<div class="clearfix">
  <div class="column menu">
    <ul>
      <li>The Flight</li>
      <li>The City</li>
      <li>The Island</li>
      <li>The Food</li>
    </ul>
  </div>
  <div class="column content">
    <h1>The City</h1>
    <p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>
    <p>You will learn more about web layout and responsive web pages in a later chapter.</p>
  </div>
</div>
<div class="footer">
  <p>Footer Text</p>
</div>
</body>
</html>

```

6) Let an image float to the right in a paragraph. Add border and margins to the image:

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
img {
  float: right;
  border: 1px dotted black;
  margin: 0px 0px 15px 20px;
}
</style>
</head>
<body>
<h2>Let an image float to the right in a paragraph</h2>
<p>In the paragraph below, the image will float to the right. A dotted black border is added to the image.  

We have also added margins to the image to push the text away from the image:  

0 px margin on the top and right side, 15 px margin on the bottom, and 20 px margin on the left side of the image.
</p>

```

</htm

7) Let an

eq:

7) Let an image with a caption float to the right:

510

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

188/313

<head>

←Style

div {

 float: right;

float: right;
width: 120px;

width: 120px,
margin: 0 0 15px 20px;

margin: 0 0 15px;
padding: 15px;

padding: 10px;
border: 1px solid black;

border: 1px solid
text-align: center;

text-align: center;

1

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

</head>

<h2>Let an image with a caption float to the right</h2>

In the paragraph below, the div element is 120 pixels wide and it contains the image. The div element will float to the right. Margins are added to the div to push the text away from the div. Borders and padding are added to the div to frame in the picture and the caption.

<div>

```
<br>CSS is fun!
```

</div>

<p>

This is some text. This is some text. This is some text.

This is some text. This is some text. This is some text.

This is some text. This is some text. This is some text.

This is some text. This is some text. This is some text.

This is some text. This is some text. This is some text.

This is some text. This is some text. This is some text.

This is some text. This is some text. This is some text.
 This is some text. This is some text. This is some text.
 This is some text. This is some text. This is some text.
 This is some text. This is some text. This is some text.
 This is some text. This is some text. This is some text.
 This is some text. This is some text. This is some text.
 This is some text. This is some text. This is some text.

</p>
</body>
</html>

8) Let the first letter of a paragraph float to the left and style the letter:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
span {
  float: left;
  width: 0.7em;
  font-size: 400%;
  font-family: algerian, courier;
  line-height: 80%;
}
</style>
</head>
<body>
<h2>Style the first letter of a paragraph and let it float left</h2>
<p>
<span>H</span>ere, the first letter of this text is embedded in a span element. The span
element has a width that is 0.7 times the size of the current font. The font-size of the span
element is 400% (quite large) and the line-height is 80%. The font of the letter in the span
will be in "Algerian".
</p>
</body>
</html>
```

9) Use float to create a homepage with a navbar, header, footer, left content and main:

content;

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}
body {
  margin: 0;
}
```

```
.header {  
background-color: #2196F3;  
color: white;  
text-align: center;  
padding: 15px;  
}  
.footer {  
background-color: #444;  
color: white;  
padding: 15px;  
}  
.topmenu {  
list-style-type: none;  
margin: 0;  
padding: 0;  
overflow: hidden;  
background-color: #777;  
}  
.topmenu li {  
float: left;  
}  
.topmenu li a {  
display: inline-block;  
color: white;  
text-align: center;  
padding: 16px;  
text-decoration: none;  
}  
.topmenu li a:hover {  
background-color: #222;  
}  
.topmenu li a.active {  
color: white;  
background-color: #4CAF50;  
}  
.column {  
float: left;  
padding: 15px;  
}  
.clearfix::after {  
content: "";  
clear: both;  
display: table;  
}  
.sidemenu {  
width: 25%;  
}  
.content {
```

```

width: 75%;
}
.sidemenu ul {
list-style-type: none;
margin: 0;
padding: 0;
}
.sidemenu li a {
margin-bottom: 4px;
display: block;
padding: 8px;
background-color: #eee;
text-decoration: none;
color: #666;
}
.sidemenu li a:hover {
background-color: #555;
color: white;
}
.sidemenu li a.active {
background-color: #008CBA;
color: white;
}

```

</style>

</head>

<body>

<ul class="topmenu">

- Home
- News
- Contact
- About

<div class="clearfix">

<div class="column sidemenu">

-
- The Flight
- The City
- The Island
- The Food
- The People
- The History
- The Oceans

</div>

<div class="column content">

<div class="header">

<h1>The City</h1>

</div>

```

<h1>Chania</h1>
<p>Chania is the capital of the Chania region on the island of Crete. The city can be
divided in two parts, the old town and the modern city.</p>
<p>You will learn more about responsive web pages in a later chapter.</p>
</div>
</div>
<div class="footer">
  <p>Footer Text</p>
</div>
</body>
</html>

```

CSS Layout - display: inline-block

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 behaviour of display: inline, display: inline-block and display: block:

eg:

```

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

```

```

border: 1px solid blue;
background-color: yellow;
}
</style>
</head>
<body>
<h1>The display Property</h1>
<h2>display: inline</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat
scelerisque elit sit amet consequat. Aliquam erat volutpat. <span class="a">Aliquam</span>
<span class="a">venenatis</span> gravida nisl sit amet facilisis. Nullam cursus fermentum
velit sed laoreet. </div>
<h2>display: inline-block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat
scelerisque elit sit amet consequat. Aliquam erat volutpat. <span class="b">Aliquam</span>
<span class="b">venenatis</span> gravida nisl sit amet facilisis. Nullam cursus fermentum
velit sed laoreet. </div>
<h2>display: block</h2>
<div>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat
scelerisque elit sit amet consequat. Aliquam erat volutpat. <span class="c">Aliquam</span>
<span class="c">venenatis</span> gravida nisl sit amet facilisis. Nullam cursus fermentum
velit sed laoreet. </div>
</body>
</html>

```

• Using inline-block to Create Navigation Links:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
.nav {
background-color: yellow;
list-style-type: none;
text-align: center;
margin: 0;
padding: 0;
}
.nav li {
display: inline-block;
font-size: 20px;
padding: 20px;
}
</style>
</head>
<body>
<h1>Horizontal Navigation Links</h1>

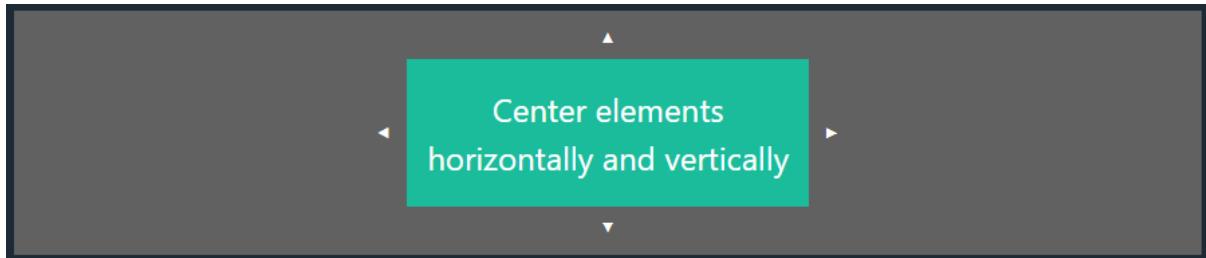
```

```

<p>By default, list items are displayed vertically. In this example we use display: inline-block to display them horizontally (side by side).</p>
<p>Note: If you resize the browser window, the links will automatically break when it becomes too crowded.</p>
<ul class="nav">
  <li><a href="#home">Home</a></li>
  <li><a href="#about">About Us</a></li>
  <li><a href="#clients">Our Clients</a></li>
  <li><a href="#contact">Contact Us</a></li>
</ul>
</body>
</html>

```

CSS Layout - Horizontal & Vertical Align



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:

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
.center {
  margin: auto;
  width: 60%;
  border: 3px solid #73AD21;
  padding: 10px;
}
</style>
</head>
<body>
<h2>Center Align Elements</h2>
<p>To horizontally center a block element (like div), use margin: auto;</p>
<div class="center">
  <p>Jashwanth<br/>Rules The<br/>World...!</p>
</div>
</body>
</html>

```

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

- Center Align Text:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
    text-align: center;
    border: 3px solid green;
}
</style>
</head>
<body>
<h2>Center Text</h2>
<div class="center">
    <p>This text is centered.</p>
</div>
</body>
</html>
```

- Center an Image:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
img {
    display: block;
    margin-left: auto;
    margin-right: auto;
}
</style>
</head>
<body>
<h2>Center an Image</h2>
<p>To center an image, set left and right margin to auto, and make it into a block element.</p>

</body>
</html>
```

- Left and Right align- Using position:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
```

```

<style>
.right {
    position: absolute;
    right: 0px;
    width: 300px;
    border: 3px solid #73AD21;
    padding: 10px;
}
</style>
</head>
<body>
<h2>Right align with the position property</h2>
<p>An example of how to right align elements with the position property:</p>
<div class="right">
    <p>In my younger and more vulnerable years my father gave me some advice that I've
    been turning over in my mind ever since.</p>
</div>
</body>
</html>
# Note: Absolute positioned elements are removed from the normal flow, and can overlap
elements.

```

- Left and Right align- Using float:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
.right {
    float: right;
    width: 300px;
    border: 3px solid #73AD21;
    padding: 10px;
}
</style>
</head>
<body>
<h2>Right align with the float property</h2>
<p>An example of how to right align elements with the float property:</p>
<div class="right">
    <p>In my younger and more vulnerable years my father gave me some advice that I've
    been turning over in my mind ever since.</p>
</div>
</body>
</html>

```

- 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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  padding: 70px 0;
  border: 3px solid green;
}
</style>
</head>
<body>
<h2>Center vertically with padding</h2>
<p>In this example, we use the padding property to center the div element vertically:</p>
<div class="center">
  <p>I am vertically centered.</p>
</div>
</body>
</html>
```

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  padding: 70px 0;
  border: 3px solid green;
  text-align: center;
}
</style>
</head>
<body>
<h2>Center with padding and text-align</h2>
<p>In this example, we use padding and text-align to center the div element both vertically and horizontally:</p>
<div class="center">
  <p>I am vertically and horizontally centered.</p>
</div>
</body>
</html>
```

- Center Vertically - Using line-height:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
```

```

<style>
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}
.center p {
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
</style>
</head>
<body>
<h2>Center with line-height</h2>
<p>In this example, we use the line-height property with a value that is equal to the height property to center the div element:</p>
<div class="center">
  <p>I am vertically and horizontally centered.</p>
</div>
</body>
</html>
• Center Vertically - Using position & transform:
If padding and line-height are not options, another solution is to use positioning and the transform property:
<!DOCTYPE html>
<html>
<head>
<style>
.center {
  height: 200px;
  position: relative;
  border: 3px solid green;
}
.center p {
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  -ms-transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
}
</style>
</head>
<body>
<h2>Center with position and transform</h2>

```

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

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

- Center Vertically - Using Flexbox:

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

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

CSS Combinators

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:

eg:

```
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

eg:

```

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:

eg:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
div + p {
```

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

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

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

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

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

```
<div>
```

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

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

```
</div>
```

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

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

```
<div>
```

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

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

```
</div>
```

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

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

```
</body>
```

```
</html>
```

- General Sibling Selector (~):

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

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div ~ p {
    background-color: yellow;
}
</style>
</head>
<body>
<h2>General Sibling Selector</h2>
<p>The general sibling selector (~) selects all elements that are next siblings of a specified element.</p>
<p>Paragraph 1.</p>
<div>
    <p>Paragraph 2.</p>
</div>
<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>
</body>
</html>
```

CSS Pseudo-Classes

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

For example, it can be used to:

Style an element when a user mouses over it

Style visited and unvisited links differently

Style an element when it gets focus

Syntax:

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

- Anchor Pseudo-Classes:

Links can be displayed in different ways:

eg:

```
/* unvisited link */
a:link {
    color: #FF0000;
}
/* visited link */
a:visited {
    color: #00FF00;
}
/* mouse over link */
```

```

a:hover {
  color: #FF00FF;
}
/* selected link */
a:active {
  color: #0000FF;
}

```

Note: a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective! a:active MUST come after a:hover in the CSS definition in order to be effective!

Pseudo-class names are not case-sensitive

- Pseudo-classes and HTML Classes:

Pseudo-classes can be combined with HTML classes

eg:

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

```

<!DOCTYPE html>
<html>
<head>
<style>
a.highlight:hover {
  color: #ff0000;
  font-size: 22px;
}
</style>
</head>
<body>
<h2>Pseudo-classes and HTML Classes</h2>
<p>When you hover over the first link below, it will change color and font size:</p>
<p><a class="highlight" href="css_syntax.asp">CSS Syntax</a></p>
<p><a href="default.asp">CSS Tutorial</a></p>
</body>
</html>

```

- Hover on <div>:

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

eg:

```

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

```

```

</head>
<body>
<p>Mouse over the div element below to change its background color:</p>
<div>Mouse Over Me</div>
</body>
</html>

```

- Simple Tooltip Hover:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}
div:hover p {
  display: block;
}
</style>
</head>
<body>
<div>Hover over this div element to show the p element
  <p>Tada! Here I am!</p>
</div>
</body>
</html>

```

- CSS - The :first-child pseudo-class:

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

~ Match the first <p> element:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
p:first-child {
  color: blue;
}
</style>
</head>
<body>
<p>This is some text.</p>
<p>This is some text.</p>

```

```
<div>
  <p>This is some text.</p>
  <p>This is some text.</p>
</div>
</body>
</html>
```

~ Match the first *i* element in all *p* elements:

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

```
<!DOCTYPE html>
<html>
<head>
<style>
p i:first-child {
  color: blue;
}
</style>
</head>
<body>
<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
</body>
</html>
```

~ Match all *i* elements in all first child *p* elements:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
p:first-child i {
  color: blue;
}
</style>
</head>
<body>
<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
<p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
<div>
  <p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
  <p>I am a <i>strong</i> person. I am a <i>strong</i> person.</p>
</div>
</body>
</html>
```

• CSS - The :lang pseudo-class:

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

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

```
<!DOCTYPE html>
```

```

<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>
<p>In this example, :lang defines the quotation marks for q elements with lang="no":</p>
</body>
</html>
• More Examples:
~ This example demonstrates how to add other styles to hyperlinks.
<!DOCTYPE html>
<html>
<head>
<style>
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;}
</style>
</head>
<body>
<h2>Styling Links</h2>
<p>Mouse over the links and watch them change layout:</p>
<p><b><a class="one" href="default.asp" target="_blank">This link changes color</a></b></p>
<p><b><a class="two" href="default.asp" target="_blank">This link changes font-size</a></b></p>

```

```

<p><b><a class="three" href="default.asp" target="_blank">This link changes
background-color</a></b></p>
<p><b><a class="four" href="default.asp" target="_blank">This link changes
font-family</a></b></p>
<p><b><a class="five" href="default.asp" target="_blank">This link changes
text-decoration</a></b></p>
</body>
</html>
~ This example demonstrates how to use the :focus pseudo-class:
<!DOCTYPE html>
<html>
<head>
<style>
input:focus {
  background-color: yellow;
}
</style>
</head>
<body>
<form action="/action_page.php" method="get">
  First name: <input type="text" name="fname"><br><br>
  Last name: <input type="text" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
• Visit This link to get all CSS Pseudo Elements and classes Reference:
https://www.w3schools.com/css/css\_pseudo\_classes.asp

```

CSS Pseudo-elements

What are Pseudo-Elements?

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

For example, it can be used to:

Style the first letter, or line, of an element

Insert content before, or after, the content of an element

Syntax:

```
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:

eg:

```
<!DOCTYPE html>
<html>
<head>
```

```

<style>
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
</style>
</head>
<body>
<p>You can use the ::first-line pseudo-element to add a special effect to the first line of a
text. Some more text. And even more, and more, and more, and more, and more, and more,
and more, and more, and more, and more, and more, and more, and more, and more.</p>
</body>
</html>

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

The following properties apply to the ::first-line pseudo-element:
font properties
color properties
background properties
word-spacing
letter-spacing
text-decoration
vertical-align
text-transform
line-height
clear

## Notice the double colon notation - ::first-line versus :first-line
The double colon replaced the single-colon notation for pseudo-elements in CSS3. This was
an attempt from W3C to distinguish between pseudo-classes and pseudo-elements.
The single-colon syntax was used for both pseudo-classes and pseudo-elements in CSS2
and CSS1.
For backward compatibility, the single-colon syntax is acceptable for CSS2 and CSS1
pseudo-elements.

• The ::first-letter Pseudo-element:
The ::first-letter pseudo-element is used to add a special style to the first letter of a text.
The following example formats the first letter of the text in all <p> elements:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
p::first-letter {
  color: #ff0000;
  font-size: xx-large;
}
</style>
</head>
<body>
<p>jashwanth</p>

```

```
</body>
</html>
# Note: The ::first-letter pseudo-element can only be applied to block-level elements.
The following properties apply to the ::first-letter pseudo- element:
font properties
color properties
background properties
margin properties
padding properties
border properties
text-decoration
vertical-align (only if "float" is "none")
text-transform
line-height
float
clear
```

- Pseudo-elements and HTML Classes:

Pseudo-elements can be combined with HTML classes:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
p.intro::first-letter {
  color: #ff0000;
  font-size: 200%;
}
</style>
</head>
<body>
<p class="intro">This is an introduction.</p>
<p>This is a paragraph with some text. A bit more text even.</p>
</body>
</html>
```

- 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:

eg:

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

```

p::first-line {
  color: #0000ff;
  font-variant: small-caps;
}
</style>
</head>
<body>
<p>You can combine the ::first-letter and ::first-line pseudo-elements to add a special effect
to the first letter and the first line of a text!</p>
</body>
</html>
• CSS - The :: before Pseudo-element:

```

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

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

eg:

```

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:

eg:

```

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

```

• CSS - The :: marker Pseudo-element:

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

The following example styles the markers of list items:

eg:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```

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

```
</style>
```

```
</head>
```

```
<body>
```

```
<ul>
```

```
  <li>Coffee</li>
```

```
  <li>Tea</li>
```

```
  <li>Milk</li>
```

```
</ul>
```

```
<ol>
```

```

<li>First</li>
<li>Second</li>
<li>Third</li>
</ol>
</body>
</html>

```

- CSS - The :: selection Pseudo-element:

The ::selection pseudo-element matches the portion of an element that is selected by a user. The following CSS properties can be applied to ::selection: color, background, cursor, and outline.

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
::selection {
  color: red;
  background: yellow;
}
</style>
</head>
<body>
<h1>Select some text on this page:</h1>
<p>This is a paragraph.</p>
<div>This is some text in a div element.</div>
</body>
</html>

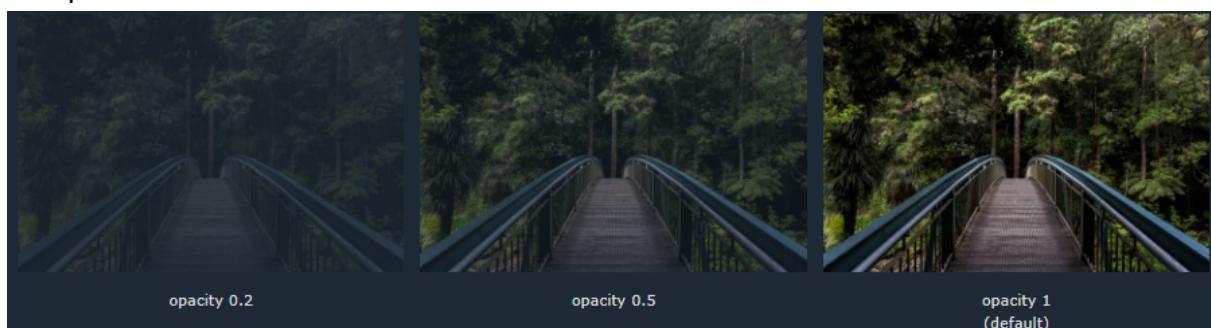
```

CSS Opacity / Transparency

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

- Transparent Image:

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



- Transparent Hover Effect:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
img {
  opacity: 0.5;
}
img:hover {
  opacity: 1.0;
}
</style>
</head>
<body>
<h1>Image Transparency</h1>
<p>The opacity property is often used together with the :hover selector to change the opacity on mouse-over:</p>



</body>
</html>

```

Example Explained:

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

An example of reversed hover effect

```

<!DOCTYPE html>
<html>
<head>
<style>
img:hover {
  opacity: 0.5;
}
</style>
</head>
<body>
<h1>Image Transparency</h1>
<p>The opacity property is often used together with the :hover selector to change the opacity on mouse-over:</p>



</body>
</html>

```

- 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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  background-color: #04AA6D;
  padding: 10px;
}
div.first {
  opacity: 0.1;
}
div.second {
  opacity: 0.3;
}
div.third {
  opacity: 0.6;
}
</style>
</head>
<body>
```

<h1>Transparent Box</h1>

<p>When using the opacity property to add transparency to the background of an element, all of its child elements become transparent as well. This can make the text inside a fully transparent element hard to read:</p>

```
<div class="first"><p>opacity 0.1</p></div>
<div class="second"><p>opacity 0.3</p></div>
<div class="third"><p>opacity 0.6</p></div>
<div><p>opacity 1 (default)</p></div>
</body>
</html>
```

- Text in Transparent Box:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div.background {
```

```

background: url(klematis.jpg) repeat;
border: 2px solid black;
}
div.transbox {
margin: 30px;
background-color: #ffffff;
border: 1px solid black;
opacity: 0.6;
}
div.transbox p {
margin: 5%;
font-weight: bold;
color: #000000;
}
</style>
</head>
<body>
<div class="background">
<div class="transbox">
<p>This is some text that is placed in the transparent box.</p>
</div>
</div>
</body>
</html>

```

Example explained

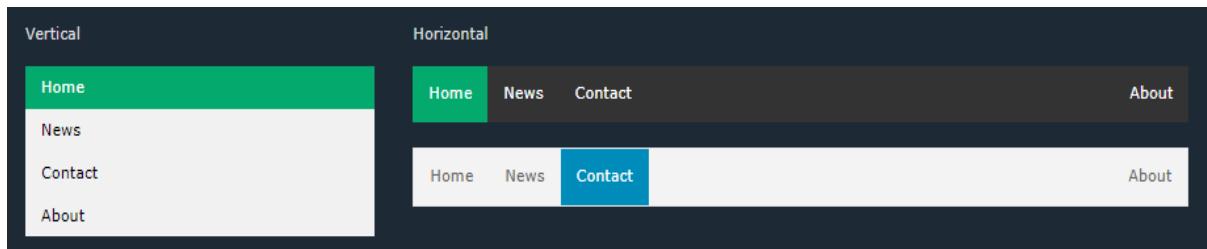
First, we create a `<div>` element (`class="background"`) with a background image, and a border.

Then we create another `<div>` (`class="transbox"`) inside the first `<div>`.

The `<div class="transbox">` have a background color, and a border - the div is transparent. Inside the transparent `<div>`, we add some text inside a `<p>` element.

CSS Navigation Bar

- Navigation Bars



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

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

Navigation Bar = List of Links

A navigation bar needs standard HTML as a base.

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

A navigation bar is basically a list of links, so using the `` and `` elements makes perfect sense:

eg:

~ First it looks like this(Without CSS):

```
<!DOCTYPE html>
<html>
<body>
<ul>
<li><a href="#home">Home</a></li>
<li><a href="#news">News</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
<p>Note: We use href="#" for test links. In a real web site this would be URLs.</p>
</body>
</html>
```

By adding some CSS to it:

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

Example explained:

`list-style-type: none;` - Removes the bullets. A navigation bar does not need list markers

`Set margin: 0;` and `padding: 0;` to remove browser default settings

The code in the example above is the standard code used in both vertical, and horizontal navigation bars, which you will learn more about in the next chapters.

• CSS Vertical Navigation Bar:

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

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

Example explained:

`display: block;` - Displaying the links as block elements makes the whole link area clickable (not just the text), and it allows us to specify the width (and padding, margin, height, etc. if you want)

`width: 60px;` - Block elements take up the full width available by default. We want to specify a 60 pixels width

You can also set the width of ``, 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:

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

```
    display: block;
}
```

~ Vertical Navigation Bar Examples:

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

```
<!DOCTYPE html>
<html>
<head>
<style>
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;
}
</style>
</head>
<body>
<h2>Vertical Navigation Bar</h2>
<ul>
    <li><a href="#home">Home</a></li>
    <li><a href="#news">News</a></li>
    <li><a href="#contact">Contact</a></li>
    <li><a href="#about">About</a></li>
</ul>
</body>
</html>
```

--> Active/Current Navigation Link:

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

```
.active {
    background-color: #04AA6D;
    color: white;
}
```

--> Center Links & Add Borders:

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

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

```
ul {
    border: 1px solid #555;
}
li {
    text-align: center;
    border-bottom: 1px solid #555;
}
li:last-child {
    border-bottom: none;
}
```

--> Full-height Fixed Vertical Navbar:

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

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    margin: 0;
}
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    width: 25%;
    background-color: #f1f1f1;
    position: fixed;
    height: 100%;
    overflow: auto;
}
li a {
    display: block;
    color: #000;
    padding: 8px 16px;
    text-decoration: none;
}
li a.active {
    background-color: #04AA6D;
    color: white;
}
li a:hover:not(.active) {
    background-color: #555;
    color: white;
}
</style>
</head>
<body>
```

```

<ul>
  <li><a class="active" href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li><a href="#about">About</a></li>
</ul>
<div style="margin-left:25%;padding:1px 16px;height:1000px;">
  <h2>Fixed Full-height Side Nav</h2>
  <h3>Try to scroll this area, and see how the sidenav sticks to the page</h3>
  <p>Notice that this div element has a left margin of 25%. This is because the side navigation is set to 25% width. If you remove the margin, the sidenav will overlay/sit on top of this div.</p>
  <p>Also notice that we have set overflow:auto to sidenav. This will add a scrollbar when the sidenav is too long (for example if it has over 50 links inside of it).</p>
  <p>Some text..</p>
  <p>Some text..</p>
</div>
</body>
</html>

```

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

CSS 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 `` elements as inline, in addition to the "standard" code from the previous example:

```

li {
  display: inline;
}

```

Example explained:

`display: inline;` - By default, `` 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 `` elements, and specify a layout for the navigation links:

```

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

```

Example explained:

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 element, to make them look good
 background-color: #dddddd; - Add a gray background-color to each element
 # Tip: Add the background-color to

 instead of each element if you want a full-width background color:
 ul {
 background-color: #dddddd;
 }
 ~ *Horizontal Navigation Bar Examples:*
 --> Create a basic horizontal navigation bar with a dark background color and change the background color of the links when the user moves the mouse over them:
 <!DOCTYPE html>
 <html>
 <head>
 <style>
 ul {
 list-style-type: none;
 margin: 0;
 padding: 0;
 overflow: hidden;
 background-color: #333;
 }
 li {
 float: left;
 }
 li a {
 display: block;
 color: white;
 text-align: center;
 padding: 14px 16px;
 text-decoration: none;
 }
 li a:hover {
 background-color: #111;
 }
 </style>
 </head>
 <body>

 Home
 News
 Contact
 About

 </body>
 </html>
 --> Active/Current Navigation Link:

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

```
.active {
    background-color: #04AA6D;
}
```

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 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:

Fixed Top

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

Fixed Bottom

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

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

--> *Gray Horizontal Navbar:*

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

```
ul {
    border: 1px solid #e7e7e7;
    background-color: #f3f3f3;
}
li a {
    color: #666;
}
```

--> *Sticky Navbar*

Add position: sticky; to 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 does not support sticky positioning. Safari requires a -webkit- prefix
(see example above). You must also specify at least one of top, right, bottom or left for sticky
positioning to work
• More Examples
~ Responsive Topnav:
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {margin: 0;}
ul.topnav {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;
}
ul.topnav li {float: left;}
ul.topnav li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
}
ul.topnav li a:hover:not(.active) {background-color: #111;}
ul.topnav li a.active {background-color: #04AA6D;}
ul.topnav li.right {float: right;}
@media screen and (max-width: 600px) {
    ul.topnav li.right,
    ul.topnav li {float: none;}
}
</style>
</head>
<body>
<ul class="topnav">
    <li><a class="active" href="#home">Home</a></li>
    <li><a href="#news">News</a></li>
    <li><a href="#contact">Contact</a></li>

```

```

<li class="right"><a href="#about">About</a></li>
</ul>
<div style="padding:0 16px;">
  <h2>Responsive Topnav Example</h2>
  <p>This example use media queries to stack the topnav vertically when the screen size is 600px or less.</p>
  <p>You will learn more about media queries and responsive web design later in our CSS Tutorial.</p>
  <h4>Resize the browser window to see the effect.</h4>
</div>
</body>
</html>
~ Responsive Sidenav:
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {margin: 0;}
ul.sidenav {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 25%;
  background-color: #f1f1f1;
  position: fixed;
  height: 100%;
  overflow: auto;
}
ul.sidenav li a {
  display: block;
  color: #000;
  padding: 8px 16px;
  text-decoration: none;
}
ul.sidenav li a.active {
  background-color: #4CAF50;
  color: white;
}
ul.sidenav li a:hover:not(.active) {
  background-color: #555;
  color: white;
}
div.content {
  margin-left: 25%;
  padding: 1px 16px;.+ 
  height: 1000px;
}

```

```

@media screen and (max-width: 900px) {
    ul.sidenav {
        width: 100%;
        height: auto;
        position: relative;
    }
    ul.sidenav li a {
        float: left;
        padding: 15px;
    }
    div.content {margin-left: 0;}
}
@media screen and (max-width: 400px) {
    ul.sidenav li a {
        text-align: center;
        float: none;
    }
}
</style>
</head>
<body>
<ul class="sidenav">
    <li><a class="active" href="#home">Home</a></li>
    <li><a href="#news">News</a></li>
    <li><a href="#contact">Contact</a></li>
    <li><a href="#about">About</a></li>
</ul>
<div class="content">
    <h2>Responsive Sidenav Example</h2>
    <p>This example use media queries to transform the sidenav to a top navigation bar when the screen size is 900px or less.</p>
    <p>We have also added a media query for screens that are 400px or less, which will vertically stack and center the navigation links.</p>
    <p>You will learn more about media queries and responsive web design later in our CSS Tutorial.</p>
    <h3>Resize the browser window to see the effect.</h3>
</div>
</body>
</html>
~ Dropdown Navbar:
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color:white;
}
ul {

```

```

list-style-type: none;
margin: 0;
padding: 0;
overflow: hidden;
background-color: #38444d;
}
li {
  float: left;
}
li a, .dropbtn {
  display: inline-block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
li a:hover, .dropdown:hover .dropbtn {
  background-color: red;
}
li.dropdown {
  display: inline-block;
}
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}
.dropdown-content a {
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
  text-align: left;
}
.dropdown-content a:hover {background-color: #f1f1f1;}
.dropdown:hover .dropdown-content {
  display: block;
}

```

</style>

</head>

<body>

Home

News

<li class="dropdown">

```

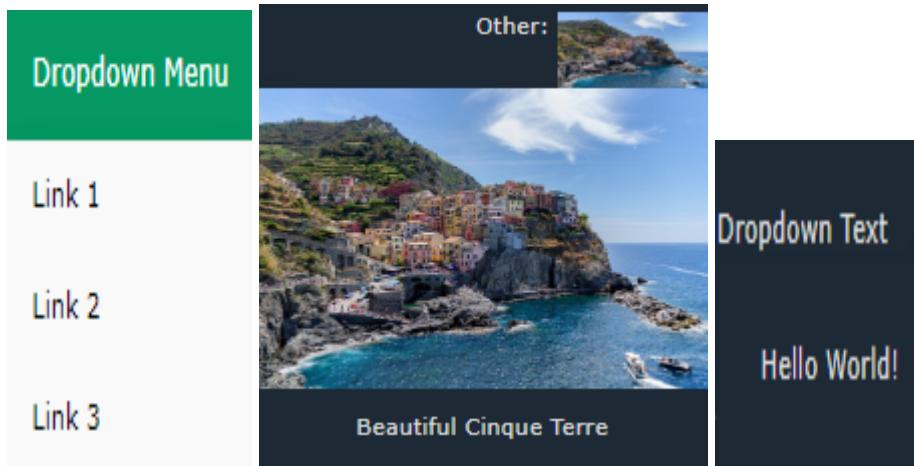
<a href="javascript:void(0)" class="dropbtn">Dropdown</a>
<div class="dropdown-content">
  <a href="#">Link 1</a>
  <a href="#">Link 2</a>
  <a href="#">Link 3</a>
</div>
</li>
</ul>
<h3>Dropdown Menu inside a Navigation Bar</h3>
<p>Hover over the "Dropdown" link to see the dropdown menu.</p>
</body>
</html>

```

CSS Dropdowns

Basic dropdown:

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



eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
.dropdown {
  position: relative;
  display: inline-block;
}
.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  padding: 12px 16px;
  z-index: 1;
}

```

```

.dropdown:hover .dropdown-content {
  display: block;
}
</style>
</head>
<body>
<h2>Hoverable Dropdown</h2>
<p>Move the mouse over the text below to open the dropdown content.</p>
<div class="dropdown">
  <span>Mouse over me</span>
  <div class="dropdown-content">
    <p>Hello World!</p>
  </div>
</div>
</body>
</html>

```

Example Explained

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

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

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

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

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

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

- [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;
}

```

- [More Examples:](#)

~ [Dropdown Image:](#)

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

```
<!DOCTYPE html>
```

```

<html>
<head>
<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">
  
  <div class="dropdown-content">
    
    <div class="desc">Beautiful Cinque Terre</div>
  </div>
</div>
</body>
</html>
~ Dropdown Navbar
How to add a dropdown menu inside a navigation bar.
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;

```

```
}

li {
  float: left;
}

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

li a:hover, .dropdown:hover .dropbtn {
  background-color: red;
}

li.dropdown {
  display: inline-block;
}

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

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

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

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

</style>
</head>
<body>
<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li class="dropdown">
    <a href="javascript:void(0)" class="dropbtn">Dropdown</a>
    <div class="dropdown-content">
      <a href="#">Link 1</a>
      <a href="#">Link 2</a>
      <a href="#">Link 3</a>
```

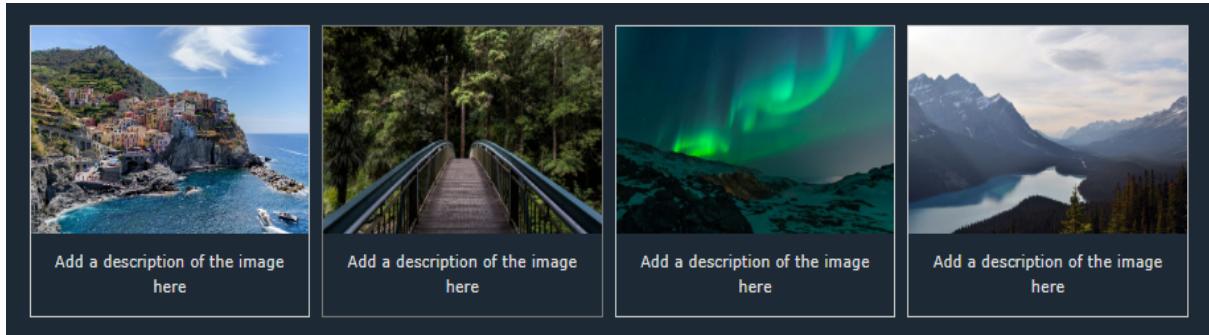
```

</div>
</li>
</ul>
<h3>Dropdown Menu inside a Navigation Bar</h3>
<p>Hover over the "Dropdown" link to see the dropdown menu.</p>
</body>
</html>

```

CSS Image Gallery

CSS can be used to create an 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">
    
    <div class="desc">
      Add a description of the image here
    </div>
  </a>
</div>
<div class="gallery">
  <a target="_blank" href="img_forest.jpg">
    
    <div class="desc">
      Add a description of the image here
    </div>
  </a>
</div>
<div class="gallery">
  <a target="_blank" href="img_northern_lights.jpg">
    
    <div class="desc">
      Add a description of the image here
    </div>
  </a>
</div>
<div class="gallery">
  <a target="_blank" href="img_mountain_lake.jpg">
    
    <div class="desc">
      Add a description of the image here
    </div>
  </a>
</div>
</body>

```

```

</a>
<div class="desc">Add a description of the image here</div>
</div>
<div class="gallery">
<a target="_blank" href="img_forest.jpg">

</a>
<div class="desc">Add a description of the image here</div>
</div>
<div class="gallery">
<a target="_blank" href="img_lights.jpg">

</a>
<div class="desc">Add a description of the image here</div>
</div>
<div class="gallery">
<a target="_blank" href="img_mountains.jpg">

</a>
<div class="desc">Add a description of the image here</div>
</div>
</body>
</html>
• More Examples:
~ Responsive Image Gallery:
<!DOCTYPE html>
<html>
<head>
<style>
div.gallery {
  border: 1px solid #ccc;
}
div.gallery:hover {
  border: 1px solid #777;
}
div.gallery img {
  width: 100%;
  height: auto;
}
div.desc {
  padding: 15px;
  text-align: center;
}
*<{
  box-sizing: border-box;
}
.responsive {
  padding: 0 6px;

```

```

float: left;
width: 24.99999%;
}
@media only screen and (max-width: 700px) {
.responsive {
width: 49.99999%;
margin: 6px 0;
}
}
@media only screen and (max-width: 500px) {
.responsive {
width: 100%;
}
}
.clearfix:after {
content: "";
display: table;
clear: both;
}
</style>
</head>
<body>
<h2>Responsive Image Gallery</h2>
<h4>Resize the browser window to see the effect.</h4>
<div class="responsive">
<div class="gallery">
<a target="_blank" href="img_5terre.jpg">

</a>
<div class="desc">Add a description of the image here</div>
</div>
</div>
<div class="responsive">
<div class="gallery">
<a target="_blank" href="img_forest.jpg">

</a>
<div class="desc">Add a description of the image here</div>
</div>
</div>
<div class="responsive">
<div class="gallery">
<a target="_blank" href="img_lights.jpg">

</a>
<div class="desc">Add a description of the image here</div>
</div>
</div>

```

```

<div class="responsive">
  <div class="gallery">
    <a target="_blank" href="img_mountains.jpg">
      
    </a>
    <div class="desc">Add a description of the image here</div>
  </div>
</div>
<div class="clearfix"></div>
<div style="padding:6px;">
  <p>This example uses media queries to rearrange the images on different screen sizes: for screens larger than 700px wide, it will show four images side by side, for screens smaller than 700px, it will show two images side by side. For screens smaller than 500px, the images will stack vertically (100%).</p>
  <p>You will learn more about media queries and responsive web design later in our CSS Tutorial.</p>
</div>
</body>
</html>

```

CSS 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.

```

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

```

Example explained:

 - 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:

eg:

```

<!DOCTYPE html>
<html>
<head>
```

```

<style>
#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;
}
</style>
</head>
<body>
<ul id="navlist">
  <li id="home"><a href="default.asp"></a></li>
  <li id="prev"><a href="css_intro.asp"></a></li>
  <li id="next"><a href="css_syntax.asp"></a></li>
</ul>
</body>
</html>
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:](#)

Image Sprites - Hover Effect

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

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

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

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

eg:

```
#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 Selectors

Style HTML Elements With Specific Attributes

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

- [CSS \[attribute\] Selector:](#)

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

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

eg:

```
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:

eg:

```
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":

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
[title~="flower"] {
  border: 5px solid yellow;
}
</style>
</head>
<body>
<h2>CSS [attribute~="value"] Selector</h2>
<p>All images with the title attribute containing the word "flower" get a yellow border.</p>



</body>
</html>
```

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".

eg:

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

- 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!

- 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":

eg:

```
[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":

eg:

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

- Styling Forms:

The attribute selectors can be useful for styling forms without class or ID:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
input[type="text"] {
    width: 150px;
    display: block;
    margin-bottom: 10px;
    background-color: yellow;
}
input[type="button"] {
    width: 120px;
    margin-left: 35px;
    display: block;
}
</style>
</head>
<body>
<h2>Styling Forms</h2>
<form name="input" action="" method="get">
    Firstname:<input type="text" name="Name" value="Peter" size="20">
    Lastname:<input type="text" name="Name" value="Griffin" size="20">
    <input type="button" value="Example Button">
</form>
</body>
</html>
```

CSS Forms

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

The form consists of three input fields:

- First Name:** A text input field with placeholder text "Your name..".
- Last Name:** A text input field with placeholder text "Your last name..".
- Country:** A dropdown menu with "USA" selected.

Below the form is a green button with the text "Try it Yourself»".

- Styling Input Fields:

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

eg:

```
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:

eg:

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

Note: that we have set the box-sizing property to border-box. This makes sure that the padding and eventually borders are included in the total width and height of the elements.

- 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;
}
```

- Coloured Inputs:

Use the background-color property to add a background color to the input, and the color property to change the text color:

eg:

```
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

eg:

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

eg:

```
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:

eg:

```
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:

```
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):

eg:

```
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:

eg:

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

- 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 */

- Responsive Form:

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}
input[type=text], select, textarea {
  width: 100%;
  padding: 12px;
  border: 1px solid #ccc;
  border-radius: 4px;
  resize: vertical;
}
label {
  padding: 12px 12px 12px 0;
  display: inline-block;
}
input[type=submit] {
  background-color: #04AA6D;
  color: white;
  padding: 12px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  float: right;
```

```

}

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

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

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

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

/* Clear floats after the columns */
.row::after {
  content: "";
  display: table;
  clear: both;
}

/* Responsive layout - when the screen is less than 600px wide, make the two columns
stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .col-25, .col-75, input[type=submit] {
    width: 100%;
    margin-top: 0;
  }
}

</style>
</head>
<body>
<h2>Responsive Form</h2>
<p>Resize the browser window to see the effect. When the screen is less than 600px wide,
make the two columns stack on top of each other instead of next to each other.</p>
<div class="container">
  <form action="/action_page.php">
    <div class="row">
      <div class="col-25">
        <label for="fname">First Name</label>
      </div>
      <div class="col-75">
        <input type="text" id="fname" name="firstname" placeholder="Your name..">
      </div>
    </div>
  </form>
</div>

```

```

</div>
<div class="row">
  <div class="col-25">
    <label for="lname">Last Name</label>
  </div>
  <div class="col-75">
    <input type="text" id="lname" name="lastname" placeholder="Your last name..">
  </div>
</div>
<div class="row">
  <div class="col-25">
    <label for="country">Country</label>
  </div>
  <div class="col-75">
    <select id="country" name="country">
      <option value="australia">Australia</option>
      <option value="canada">Canada</option>
      <option value="usa">USA</option>
    </select>
  </div>
</div>
<div class="row">
  <div class="col-25">
    <label for="subject">Subject</label>
  </div>
  <div class="col-75">
    <textarea id="subject" name="subject" placeholder="Write something.."
style="height:200px"></textarea>
  </div>
</div>
<br>
<div class="row">
  <input type="submit" value="Submit">
</div>
</form>
</div>
</body>
</html>

```

CSS Counters

CSS counters are "variables" maintained by CSS whose values can be incremented by CSS rules (to track how many times they are used). Counters let you adjust the appearance of content based on its placement in the document.

- [Automatic Numbering With Counters:](#)

CSS counters are like "variables". The variable values can be incremented by CSS rules (which will track how many times they are used).

To work with CSS counters we will use the following properties:

counter-reset - Creates or resets a counter

counter-increment - Increments a counter value

content - Inserts generated content

counter() or counters() function - Adds the value of a counter to an element

To use a CSS counter, it must first be created with counter-reset.

The following example creates a counter for the page (in the body selector), then increments the counter value for each `<h2>` element and adds "Section <value of the counter>:" to the beginning of each `<h2>` element:

eg:

```
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>":

eg:

```
body {
  counter-reset: section;
}

h1 {
  counter-reset: subsection;
}

h1::before {
  counter-increment: section;
  content: "Section " counter(section) ". ";
}

h2::before {
  counter-increment: subsection;
  content: counter(section) "." counter(subsection) " ";
}
```

A counter can also be useful to make outlined lists because a new instance of a counter is automatically created in child elements. Here we use the `counters()` function to insert a string between different levels of nested counters:

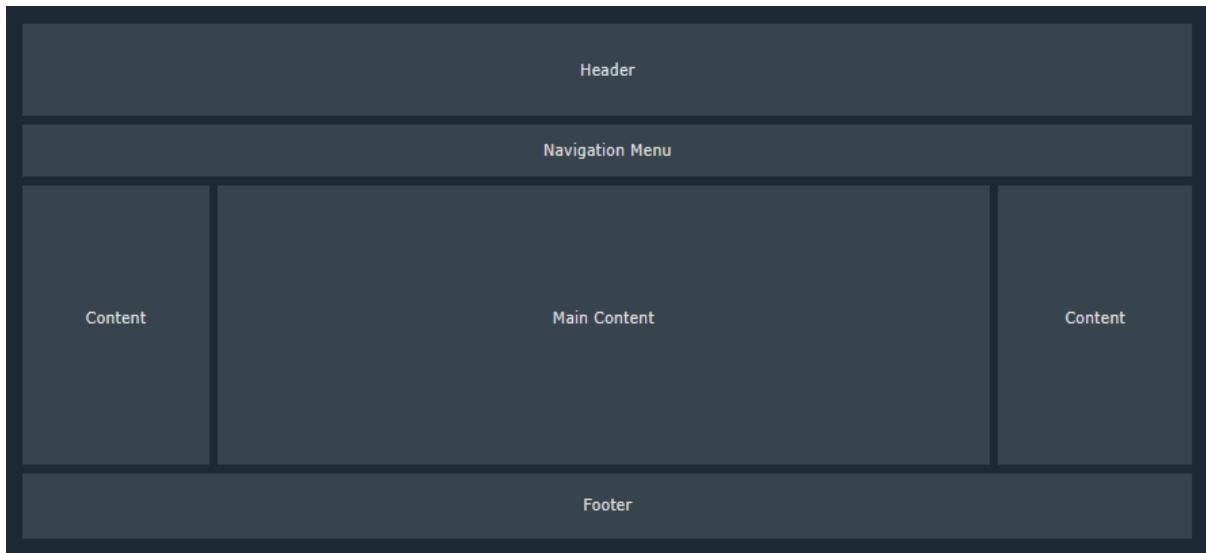
eg:

```
ol {
  counter-reset: section;
  list-style-type: none;
}

li::before {
  counter-increment: section; content: counters(section,".") " "; }
```

CSS Website Layout

A website is often divided into headers, menus, content and a footer:



There are tons of different layout designs to choose from. However, the structure above is one of the most common ones.

- 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:

eg:

```
.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:

eg:

```
/* 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; }
```

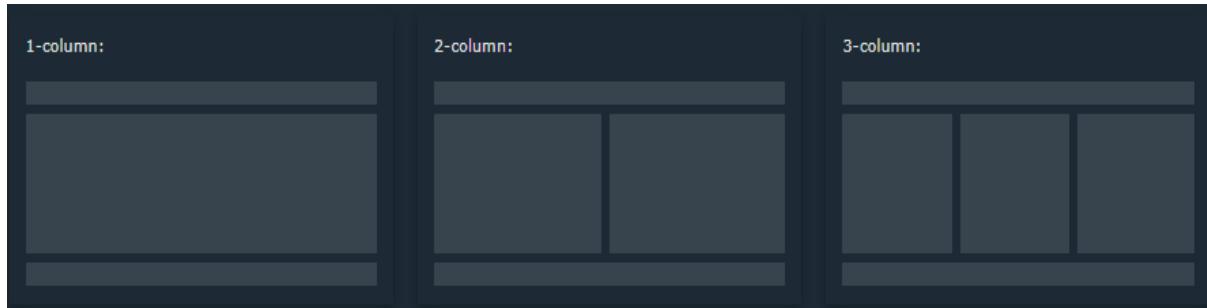
- Content:

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:

eg:

```
/* 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.

- 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:

eg:

```
.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:

eg:

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

- Responsive Website Layout:

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}
body {
  font-family: Arial;
  padding: 10px;
  background: #f1f1f1;
}
/* Header/Blog Title */
.header {
  padding: 30px;
  text-align: center;
  background: white;
}
.header h1 {
  font-size: 50px;
```

```
}

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

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

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

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

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

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

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

/* Clear floats after the columns */
.row::after {
  content: "";
}
```

```

display: table;
clear: both;
}
/* Footer */
.footer {
padding: 20px;
text-align: center;
background: #ddd;
margin-top: 20px;
}
/* Responsive layout - when the screen is less than 800px wide, make the two columns
stack on top of each other instead of next to each other */
@media screen and (max-width: 800px) {
.leftcolumn, .rightcolumn {
width: 100%;
padding: 0;
}
}
/* Responsive layout - when the screen is less than 400px wide, make the navigation links
stack on top of each other instead of next to each other */
@media screen and (max-width: 400px) {
.topnav a {
float: none;
width: 100%;
}
}
</style>
</head>
<body>
<div class="header">
<h1>My Website</h1>
<p>Resize the browser window to see the effect.</p>
</div>
<div class="topnav">
<a href="#">Link</a>
<a href="#">Link</a>
<a href="#">Link</a>
<a href="#" style="float:right">Link</a>
</div>
<div class="row">
<div class="leftcolumn">
<div class="card">
<h2>TITLE HEADING</h2>
<h5>Title description, Dec 7, 2017</h5>
<div class="fakeimg" style="height:200px;">Image</div>
<p>Some text..</p>

```

```

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

```

CSS Units

CSS has several different units for expressing a length.

Many CSS properties take "length" values, such as width, margin, padding, font-size, etc.

Length is a number followed by a length unit, such as 10px, 2em, etc.

eg:

Set different length values, using px (pixels):

```

h1 {
  font-size: 60px;
}

```

```
p {
  font-size: 25px;
  line-height: 50px;
}
```

Note: A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted.

For some CSS properties, negative lengths are allowed.

There are two types of length units: absolute and relative.

- Absolute Lengths:

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

Unit	Description
cm	centimetres.
mm	millimetres.
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 the width of the "0" (zero).
rem	Relative to the 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 a perfectly scalable layout!

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

CSS 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.

Look at the following examples:

eg 1:

In this example, we have used the "p" element as selector, and specified a red color for this element. The text will be red:

```
<html>
<head>
  <style>
    p {color: red;}
  </style>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
```

Now, look at example 2:

eg 2:

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>
```

Now, look at example 3:

eg 3:

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

Now, look at example 4:

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

- Specificity Hierarchy:

Every CSS selector has its place in the specificity hierarchy.

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

Inline styles - Example: `<h1 style="color: pink;">`

IDs - Example: `#navbar`

Classes, pseudo-classes, attribute selectors - Example: `.test, :hover, [href]`

Elements and pseudo-elements - Example: `h1, ::before`

- How to calculate Specificity ?:

Memorise 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 1: 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 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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {background-color: yellow;}
h1 {background-color: red;}
</style>
</head>
<body>
<h1>This is heading 1</h1>
</body>
</html>
```

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div#a {background-color: green;}
#a {background-color: yellow;}
div[id=a] {background-color: blue;}
</style>
</head>
<body>
<div id="a">This is a div</div>
</body>
</html>
```

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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.intro {background-color: yellow;}
h1 {background-color: red;}
</style>
</head>
<body>
<h1 class="intro">This is a heading</h1>
</body>
</html>
```

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

CSS The !important Rule

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!

Let us look at an example:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
#myid {
    background-color: blue;
}
.myclass {
    background-color: gray;
}
p {
    background-color: red !important;
}
</style>
</head>
<body>
<p>This is some text in a paragraph.</p>
<p class="myclass">This is some text in a paragraph.</p>
```

```
<p id="myid">This is some text in a paragraph.</p>
</body>
</html>
```

Example Explained

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:

eg:

```
#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:

eg:

```
.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:

eg:

```
.button {
    background-color: #8c8c8c;
    color: white;
    padding: 5px;
```

```

border: 1px solid black;
}
#myDiv a {
  color: red;
  background-color: yellow;
}

```

To "force" all buttons to have the same look, no matter what, we can add the !important rule to the properties of the button, like this:

eg:

```

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

```

CSS Math Functions

The CSS math functions allow mathematical expressions to be used as property values. Here, we will discuss the calc(), max() and min() functions.

- The calc() Function:

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

calc Syntax:

calc(expression)

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
#div1 {
  position: absolute;
  left: 50px;
  width: calc(100% - 100px);
  border: 1px solid black;
  background-color: yellow;
  padding: 5px;
}
</style>
</head>

```

```

<body>
<h1>The calc() Function</h1>
<p>Create a div that stretches across the window, with a 50px gap between both sides of
the div and the edges of the window:</p>
<div id="div1">Some text...</div>
</body>
</html>

```

- The max() Function:

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

max Syntax:

max(value1, value2, ...)

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

eg:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

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

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>The max() Function</h1>
```

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

```
<div id="div1">Some text...</div>
```

<p>Resize the browser window to see the effect.</p>

```
</body>
```

```
</html>
```

- The min() Function:

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

min Syntax:

min(value1, value2, ...)

Value	Description
-------	-------------

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

* * * * *

CSS ADVANCED

CSS Rounded Corners

With the CSS border-radius property, you can give any element "rounded corners".

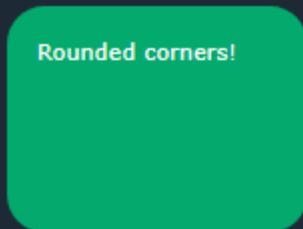
CSS Border-radius Property:

The CSS border-radius property defines the radius of an element's corners.

Tip: This property allows you to add rounded corners to elements!

Here are three examples:

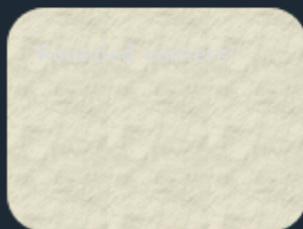
1. Rounded corners for an element with a specified background color:



2. Rounded corners for an element with a border:



3. Rounded corners for an element with a background image:



eg:

Here is the code:

```
#rcorners1 {  
    border-radius: 25px;  
    background: #73AD21;  
    padding: 20px;  
    width: 200px;  
    height: 150px;  
}  
  
#rcorners2 {  
    border-radius: 25px;  
    border: 2px solid #73AD21;
```

```

padding: 20px;
width: 200px;
height: 150px;
}
#rcorners3 {
border-radius: 25px;
background: url(paper.gif);
background-position: left top;
background-repeat: repeat;
padding: 20px;
width: 200px;
height: 150px;
}

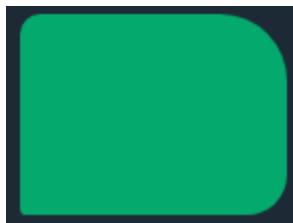
```

Tip: The border-radius property is actually a shorthand property for the border-top-left-radius, border-top-right-radius, border-bottom-right-radius and border-bottom-left-radius properties

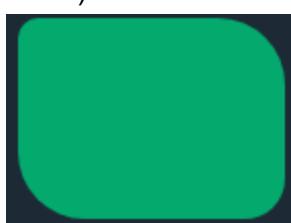
- CSS border-radius - Specify Each Corner:

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

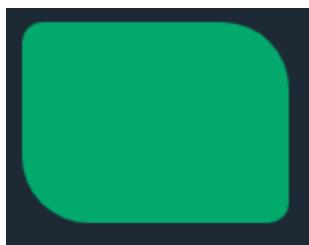
~ Four values - border-radius: 15px 50px 30px 5px; (first value applies to top-left corner, second value applies to top-right corner, third value applies to bottom-right corner, and fourth value applies to bottom-left corner):



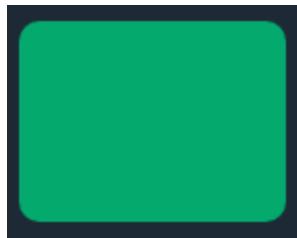
~ Three values - border-radius: 15px 50px 30px; (first value applies to top-left corner, second value applies to top-right and bottom-left corners, and third value applies to bottom-right corner):



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



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



Here is the code:

eg:

```
#rcorners1 {
    border-radius: 15px 50px 30px 5px;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}
#rcorners2 {
    border-radius: 15px 50px 30px;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}
#rcorners3 {
    border-radius: 15px 50px;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}
#rcorners4 {
    border-radius: 15px;
    background: #73AD21;
    padding: 20px;
    width: 200px;
    height: 150px;
}
~ You could create Elliptical corners:
<!DOCTYPE html>
<html>
<head>
<style>
#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;
}
</style>
</head>
<body>
<h1>The border-radius Property</h1>
<p>Elliptical border - border-radius: 50px / 15px:</p>
<p id="rcorners1"></p>
<p>Elliptical border - border-radius: 15px / 50px:</p>
<p id="rcorners2"></p>
<p>Ellipse border - border-radius: 50%:</p>
<p id="rcorners3"></p>
</body>
</html>

```

CSS Border Images

With the CSS border-image property, you can set an image to be used as the border around an element.

- 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.

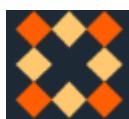
The property has three parts:

The image to use as the border

Where to slice the image

Define whether the middle sections should be repeated or stretched.

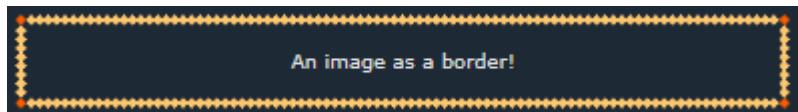
We will use the following image (called "border.png"):



The border-image property takes the image and slices it into nine sections, like a tic-tac-toe board. It then places the corners at the corners, and the middle sections are repeated or stretched as you specify.

Note: For border-image to work, the element also needs the border property set!

Here, the middle sections of the image are repeated to create the border:



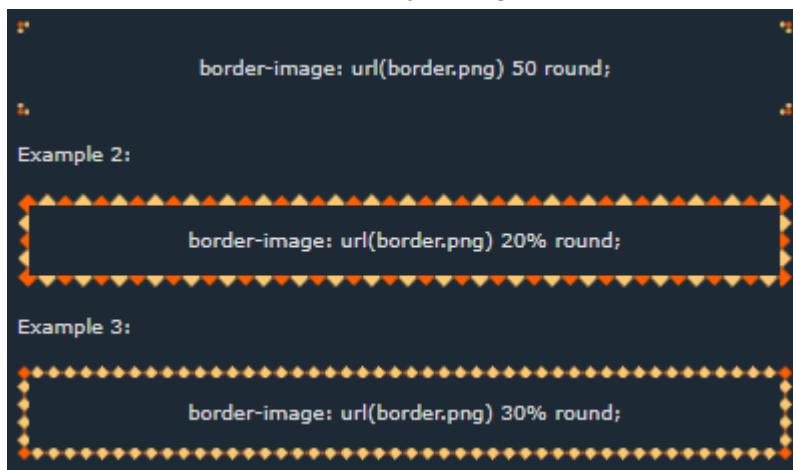
here is the code:

```
#borderimg {
    border: 10px solid transparent;
    padding: 15px;
    border-image: url(border.png) 30 round;
}

# Tip: The border-image property is actually a shorthand property for the border-image-source, border-image-slice, border-image-width, border-image-outset and border-image-repeat properties.
```

- CSS Border-image - Different Slice Values:

Different slice values completely changes the look of the border:



Here is the code:

```
#borderimg1 {
    border: 10px solid transparent;
    padding: 15px;
    border-image: url(border.png) 50 round;
}

#borderimg2 {
    border: 10px solid transparent;
    padding: 15px;
    border-image: url(border.png) 20% round;
}

#borderimg3 {
    border: 10px solid transparent;
    padding: 15px;
    border-image: url(border.png) 30% round;
}
```

- CSS Border Image Properties:

Property	Description
border-image	A shorthand property for setting all the border-image-* properties
border-image-source	Specifies the path to the image to be used as a border.
border-image-slice	Specifies how to slice the border image.
border-image-width	Specifies the widths of the border image.
border-image-outset	Specifies the amount by which the border image area extends beyond the border box.
border-image-repeat	Specifies whether the border image should be repeated, rounded or stretched .

CSS Multiple Backgrounds

CSS 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):



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):

eg:

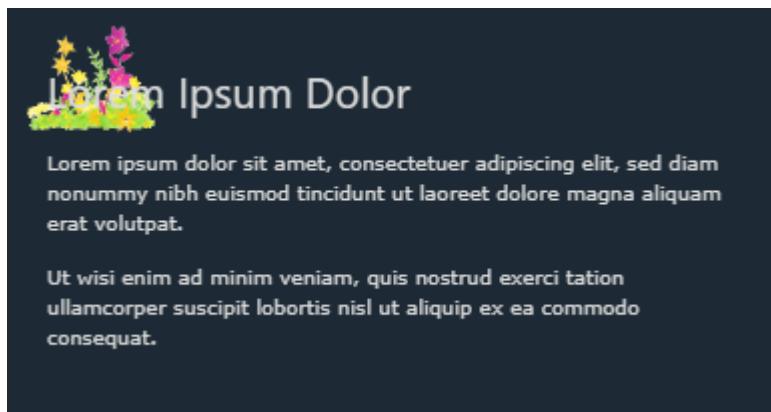
```
#example1 {
    background: url(img_flwr.gif) right bottom no-repeat, url(paper.gif) left top repeat;
}
```

- CSS Background Size:

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

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

The following example resizes a background image to much smaller than the original image (using pixels):



eg:

```
#div1 {
    background: url(img_flower.jpg);
    background-size: 100px 80px;
    background-repeat: no-repeat;
}
```

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

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

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

The following example illustrates the use of contain and cover:

eg:

```
#div1 {
    background: url(img_flower.jpg);
    background-size: contain;
    background-repeat: no-repeat;
}
#div2 {
    background: url(img_flower.jpg);
    background-size: cover;
    background-repeat: no-repeat;
}
```

The background-size Property

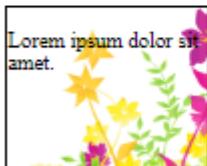
background-size: contain:



background-size: cover:



No background-size defined:



Original image:



- 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:

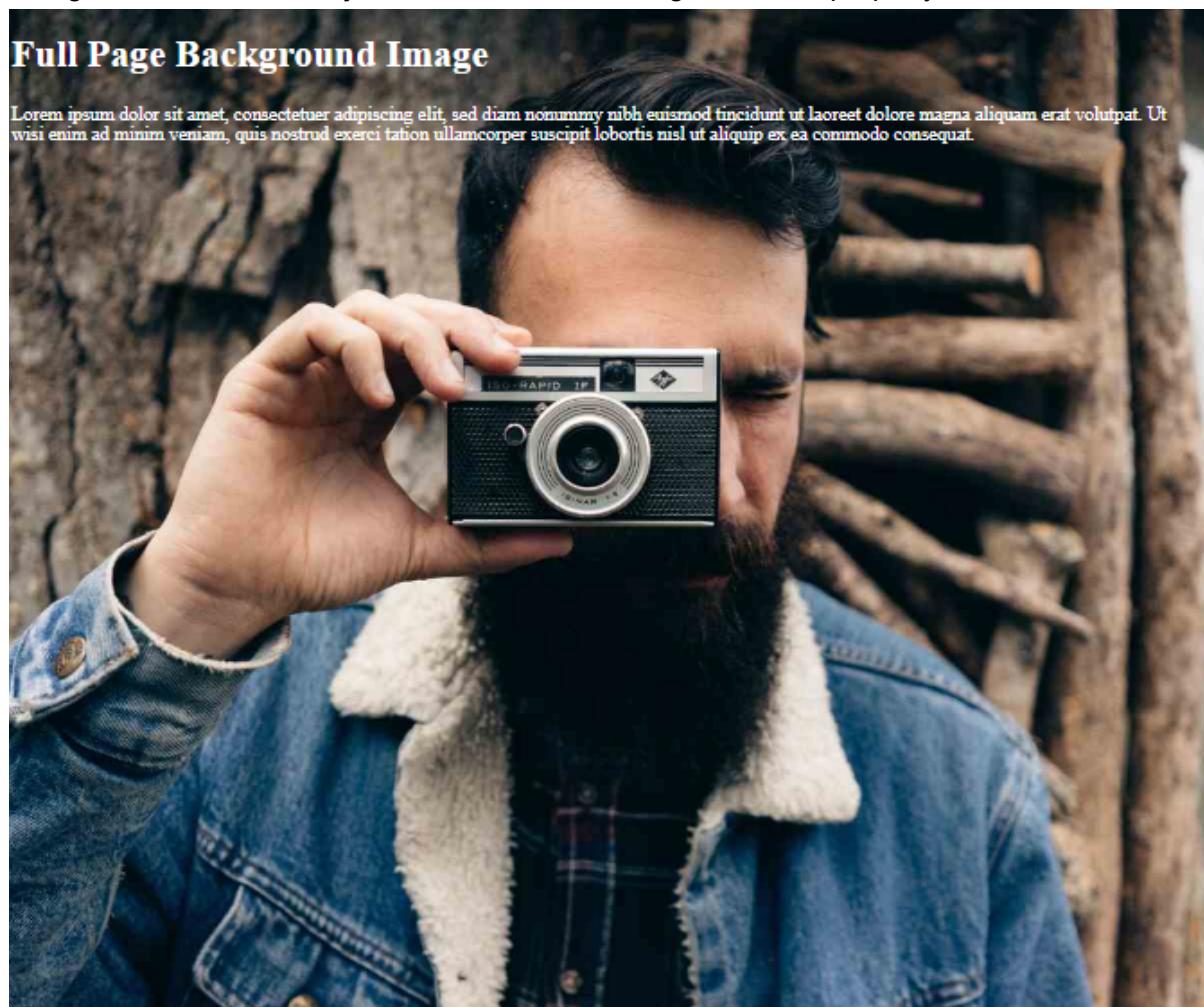
Fill the entire page with the image (no white space)

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:



eg:

```
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.



Page content..

Note that this technique will also make the image responsive. Resize the browser window to see the effect.

eg:

```
.hero-image {
  background: url(img_man.jpg) no-repeat center;
  background-size: cover;
  height: 500px;
  position: relative;
}
```

- CSS background-origin Property:

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

The property takes three different values:

border-box - the background image starts from the upper left corner of the border

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

content-box - the background image starts from the upper left corner of the content

The following example illustrates the background-origin property:

```
#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:

eg:

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

CSS Color Keywords

- 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.

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-image: url("paper.gif");
}
div.ex1 {
    background-color: lightgreen;
    border: 2px solid black;
    padding: 15px;
}
div.ex2 {
    background-color: transparent;
    border: 2px solid black;
    padding: 15px;
}
</style>
</head>
<body>
<h2>The transparent Keyword</h2>
<div class="ex1">This div has a light green background.</div>
<br>
<div class="ex2">This div has a transparent background.</div>
</body>
</html>
```

Note: The transparent keyword is equivalent to rgba(0,0,0,0). RGBA color values are an extension of RGB color values with an alpha channel - which specifies the opacity for a color

- 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.

eg 1:

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;
}
```

eg 2:

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;
}
```

eg 3:

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.

eg:

In this example the 's border settings will be inherited from the parent element:

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

CSS Gradients

Gradient Backgrounds

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:

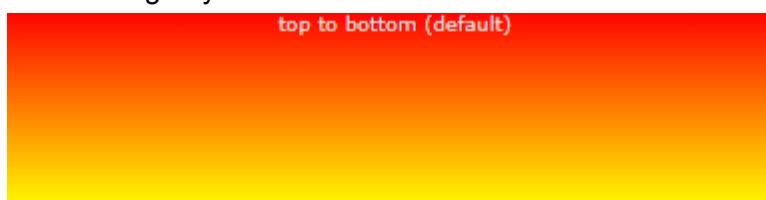
To create a linear gradient you must define at least two color stops. Color stops are the colors you want to render smooth transitions among. You can also set a starting point and a direction (or an angle) along with the gradient effect.

Syntax:

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

~ Direction - Top to Bottom (This is default):

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



eg:

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

~ Direction - Left to Right:

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



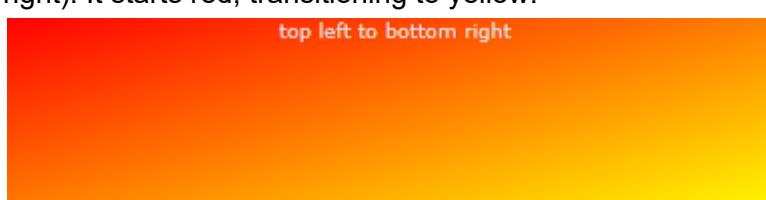
eg:

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

~ Direction - Diagonal:

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

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



eg:

```
#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".

Syntax:

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

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



eg:

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

~~ Using Multiple Color Stops:

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



eg:

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

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



eg:

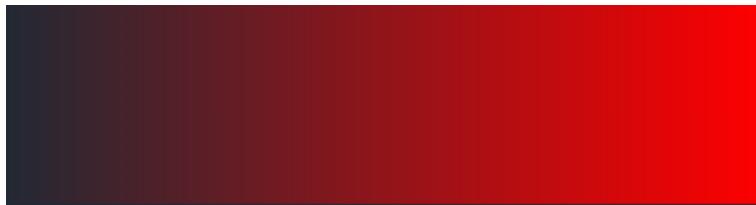
```
#grad {
  background-image: linear-gradient(to right, red, orange, yellow, green, blue, indigo, violet);
}
```

~~ Using Transparency:

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

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

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

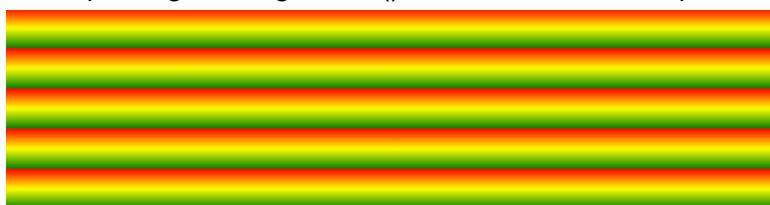


eg:

```
#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:



eg:

A repeating linear gradient:

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

• CSS Radial Gradients:

A radial gradient is defined by its center.

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

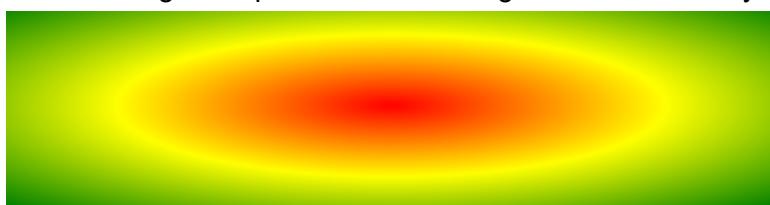
Syntax:

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

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

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

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

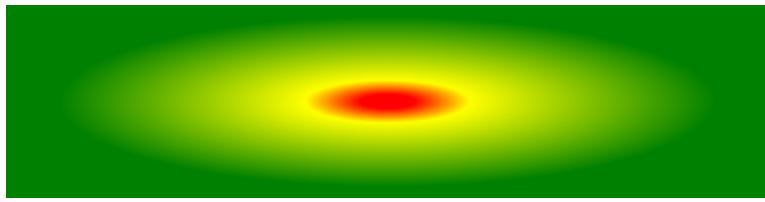


eg:

```
#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:



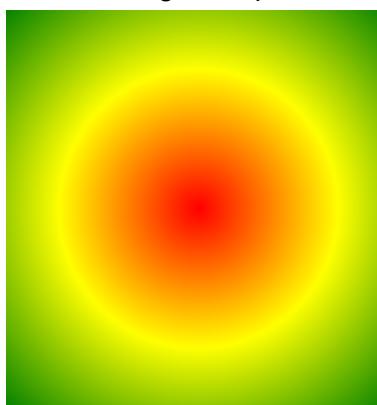
eg:

```
#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:



eg:

```
#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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
#grad1 {
  height: 150px;
  width: 150px;
  background-color: red; /* For browsers that do not support gradients */
  background-image: radial-gradient(closest-side at 60% 55%, red, yellow, black);
}
#grad2 {
  height: 150px;
  width: 150px;
```

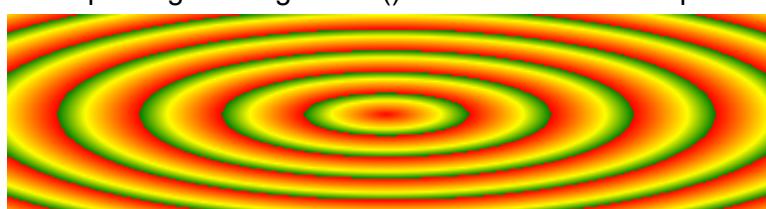
```

background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(farthest-side at 60% 55%, red, yellow, black);
}
#grad3 {
height: 150px;
width: 150px;
background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(closest-corner at 60% 55%, red, yellow, black);
}
#grad4 {
height: 150px;
width: 150px;
background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(farthest-corner at 60% 55%, red, yellow, black);
}
</style>
</head>
<body>
<h1>Radial Gradients - Different size keywords</h1>
<h2>closest-side:</h2>
<div id="grad1"></div>
<h2>farthest-side:</h2>
<div id="grad2"></div>
<h2>closest-corner:</h2>
<div id="grad3"></div>
<h2>farthest-corner (default):</h2>
<div id="grad4"></div>
</body>
</html>

```

~ ~ Repeating a radial - Gradient:

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



eg:

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

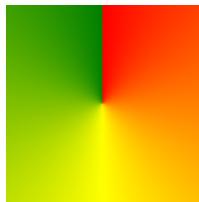
```
background-image: conic-gradient([from angle] [at position,] color [degree], color [degree],
...);
```

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:

The following example shows a conic gradient with three colors:



eg:

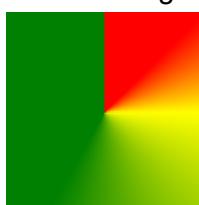
A conic gradient with three colors:

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

Note: We can give 'n' Number of colors to the gradient.

~ Conic Gradient: Three Colors and Degrees:

The following example shows a conic gradient with three colors and a degree for each color:



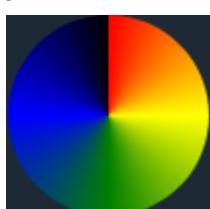
eg:

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

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

~ Create a Pie chart:

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



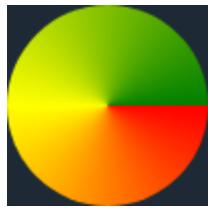
eg:

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



eg:

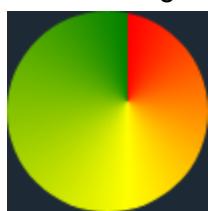
A conic gradient with a from angle:

```
#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%:



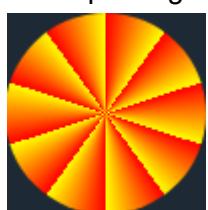
eg:

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:



eg:

A repeating conic gradient:

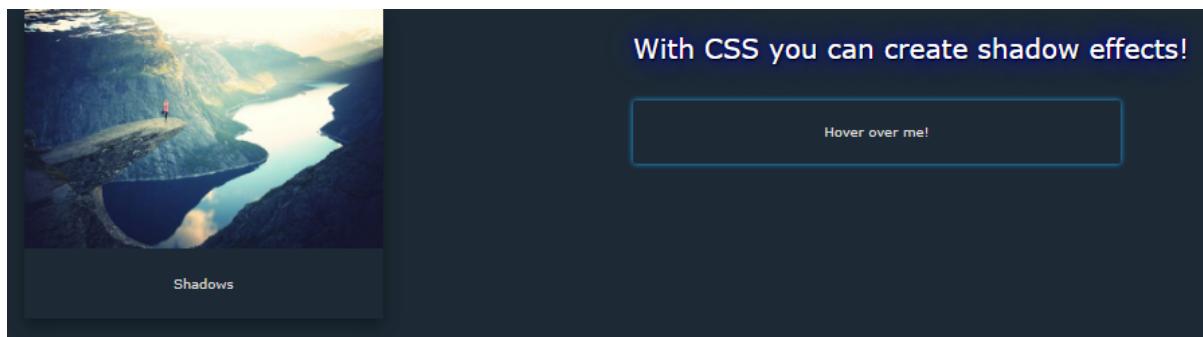
```
#grad {
  background-image: repeating-conic-gradient(red 10%, yellow 20%);
  border-radius: 50%;
}
```

~ ~ CSS Gradient Functions:

Function	Description
<code>conic-gradient()</code>	Creates a conic gradient. Define at least two colors (around a center point)
<code>linear-gradient()</code>	Creates a linear gradient. Define at least two colors (top to bottom)
<code>radial-gradient()</code>	Creates a radial gradient. Define at least two colors (center to edges)
<code>repeating-conic-gradient()</code>	Repeats a conic gradient
<code>repeating-linear-gradient()</code>	Repeats a linear gradient
<code>repeating-radial-gradient()</code>	Repeats a radial gradient

CSS Shadows

- CSS Text Shadows:



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):

eg:

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

- Next add a color to the text:

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

- Then, add a blur effect to the shadow:

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

eg:

```
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;  
}
```

- Border Around Text:

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

- CSS Box Shadow:

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.

eg:

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

~ *Specify a Color for the Shadow:*

The color parameter defines the color of the shadow.

eg:

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

~ *Add a Blur Effect to the Shadow:*

The blur parameter defines the blur radius. The higher the number, the more blurred the shadow will be.

eg:

```
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.

eg:

```
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.

eg:

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

~ Add Multiple Shadows:

An element can also have multiple shadows:

eg:

```
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:

- Text Cards:

```
<!DOCTYPE html>
<html>
<head>
<style>
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;
}
div.header {
  background-color: #4CAF50;
  color: white;
  padding: 10px;
  font-size: 40px;
}
div.container {
  padding: 10px;
}
</style>
</head>
<body>
<h1>Create Cards</h1>
<p>The box-shadow property can be used to create paper-like cards:</p>
<div class="card">
  <div class="header">
    <h1>1</h1>
  </div>
  <div class="container">
    <p>January 1, 2021</p>
  </div>
</div>
</body>
```

```

</html>
- Image Cards:
<!DOCTYPE html>
<html>
<head>
<style>
div.polaroid {
  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;
}
div.container {
  padding: 10px;
}
</style>
</head>
<body>
<h1>Create Polaroid Images</h1>
<p>The box-shadow property can be used to create polaroid images:</p>
<div class="polaroid">
  
  <div class="container">
    <p>Hardanger, Norway</p>
  </div>
</div>
</body>
</html>

```

CSS Text Effects

CSS Text Overflow, Word Wrap, Line Breaking Rules, and Writing Modes

- CSS Text - Overflow:

The CSS text-overflow property specifies how overflowed content that is not displayed should be signaled to the user.

It can be clipped:

This is some long text that will not fit in the box

or it can be rendered as an ellipsis (...):

This is some long text that...

eg:

```

p.test1 {
  white-space: nowrap;
  width: 200px;
  border: 1px solid #000000;
  overflow: hidden;
  text-overflow: clip;
}
p.test2 {

```

```

white-space: nowrap;
width: 200px;
border: 1px solid #000000;
overflow: hidden;
text-overflow: ellipsis;
}

```

The following example shows how you can display the overflowed content when hovering over the element:

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
div.test {
    white-space: nowrap;
    width: 200px;
    overflow: hidden;
    border: 1px solid #000000;
}
div.test:hover {
    overflow: visible;
}
</style>
</head>
<body>
<p>Hover over the two divs below, to see the entire text.</p>
<div class="test" style="text-overflow:ellipsis;">This is some long text that will not fit in the
box</div>
<br>
<div class="test" style="text-overflow:clip;">This is some long text that will not fit in the
box</div>
</body>
</html>

```

- CSS Word Wrapping:

The CSS word-wrap property allows long words to be able to be broken and wrap onto the next line.

If a word is too long to fit within an area, it expands outside:

This paragraph
contains a very long
word: thisisaveryveryveryveryverylongword

. The long word will
break and wrap to
the next line.

~ The word-wrap property allows you to force the text to wrap - even if it means splitting it in the middle of a word:

This paragraph
contains a very long
word: thisisaveryver

yveryve
 ryverylongword
 . The long word will
 break and wrap to
 the next line.

The CSS code is as follows:

Allow long words to be able to be broken and wrap onto the next line:

eg:

```
p {  

  word-wrap: break-word;  

}
```

- CSS Word Break:

The CSS word-break property specifies line breaking rules.

This paragraph
 contains some
 text. This line
 will-break-at-
 hyphens

This paragraph co
 ntains some text.

The lines will brea
 k at any characte
 r.

eg:

```
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.

The following example shows some different writing modes:

eg:

```
<!DOCTYPE html>  

<html>  

<head>  

<style>  

p.test1 {  

  writing-mode: horizontal-tb;  

}  

span.test2 {  

  writing-mode: vertical-rl;  

}  

p.test2 {  

  writing-mode: vertical-rl;
```

```

}
</style>
</head>
<body>
<h1>The writing-mode Property</h1>
<p class="test1">Some text with default writing-mode.</p>
<p>Some text with a span element with a <span class="test2">vertical-rl</span>
writing-mode.</p>
<p class="test2">Some text with writing-mode: vertical-rl.</p>
</body>
</html>

```

- CSS Text Effect Properties:

Property	Description
text-justify	Specifies how justified text should be aligned and spaced
text-overflow	Specifies how overflowed content that is not displayed should be signaled to the user
word-break	Specifies line breaking rules for non-CJK scripts
word-wrap	Allows long words to be able to be broken and wrap onto the next line
writing-mode	Specifies whether lines of text are laid out horizontally or vertically

CSS Web Fonts

- The CSS @font-face Rule:

Web fonts allow Web designers to use fonts that are not installed on the user's computer. When you have found/bought the font you wish to use, just include the font file on your web server, and it will be automatically downloaded to the user when needed.

Your "own" fonts are defined within the CSS @font-face rule.

- Different Font Formats:

~ TrueType Fonts (TTF)

TrueType is a font standard developed in the late 1980s, by Apple and Microsoft. TrueType is the most common font format for both the Mac OS and Microsoft Windows operating systems.

~ OpenType Fonts (OTF)

OpenType is a format for scalable computer fonts. It was built on TrueType, and is a registered trademark of Microsoft. OpenType fonts are used commonly today on the major computer platforms.

~ The Web Open Font Format (WOFF)

WOFF is a font format for use in web pages. It was developed in 2009, and is now a W3C Recommendation. WOFF is essentially OpenType or TrueType with compression and additional metadata. The goal is to support font distribution from a server to a client over a network with bandwidth constraints.

~ The Web Open Font Format (WOFF 2.0)

TrueType/OpenType font that provides better compression than WOFF 1.0.

~ SVG Fonts/Shapes

SVG fonts allow SVG to be used as glyphs when displaying text. The SVG 1.1 specification define a font module that allows the creation of fonts within an SVG document. You can also

apply CSS to SVG documents, and the @font-face rule can be applied to text in SVG documents.

~ Embedded OpenType Fonts (EOT)

EOT fonts are a compact form of OpenType fonts designed by Microsoft for use as embedded fonts on web pages.

- 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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
@font-face {
    font-family: myFirstFont;
    src: url(sansation_light.woff);
}
* {
    font-family: myFirstFont;
}
</style>
</head>
<body>
<h1>The @font-face Rule</h1>
<div>
```

With CSS, websites can use fonts other than the pre-selected "web-safe" fonts.

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

- Using Bold Text:

You must add another @font-face rule containing descriptors for bold text:

eg:

```
@font-face {
    font-family: myFirstFont;
    src: url(sansation_bold.woff);
    font-weight: bold;
}
```

The file "sansation_bold.woff" is another font file, that contains the bold characters for the Sansation font.

Browsers will use this whenever a piece of text with the font-family "myFirstFont" should render as bold.

This way you can have many @font-face rules for the same font.

- CSS Font Descriptors:

The following table lists all the font descriptors that can be defined inside the @font-face rule:

Descriptor	Values	Description
font-family	<i>name</i>	Required. Defines a name for the font
src	<i>URL</i>	Required. Defines the URL of the font file
font-stretch	normal condensed ultra-condensed extra-condensed semi-condensed expanded semi-expanded extra-expanded ultra-expanded	Optional. Defines how the font should be stretched. Default is "normal"
font-style	normal italic oblique	Optional. Defines how the font should be styled. Default is "normal"
font-weight	normal bold 100 200 300 400 500 600 700 800 900	Optional. Defines the boldness of the font. Default is "normal"
unicode-range	<i>unicode-range</i>	Optional. Defines the range of UNICODE characters the font supports. Default is "U+0-10FFFF"

CSS 2D Transforms

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

- CSS 2D Transform Methods:

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

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

~ *The translate() Method:*

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

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
    transform: translate(50px,100px);
}
</style>
</head>
<body>
<h1>The translate() Method</h1>
<p>The translate() method moves an element from its current position:</p>
<div>
This div element is moved 50 pixels to the right, and 100 pixels down from its current
position.
</div>
</body>
</html>

~ The Rotate() Method:
The rotate() method rotates an element clockwise or counter-clockwise according to a given
degree.
The following example rotates the <div> element clockwise with 20 degrees:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 300px;
    height: 100px;
    background-color: yellow;
    border: 1px solid black;
}
div#myDiv {
    transform: rotate(20deg);
}
</style>
</head>
<body>
<h1>The rotate() Method</h1>
<p>The rotate() method rotates an element clockwise or counter-clockwise.</p>
<div>
This is a normal div element.

```

```
</div>
<div id="myDiv">
This div element is rotated clockwise 20 degrees.
</div>
</body>
</html>
```

Using negative values will rotate the element counter-clockwise.

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

eg:

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

~ The Scale() Method:

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

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  margin: 150px;
  width: 200px;
  height: 100px;
  background-color: yellow;
  border: 1px solid black;
  transform: scale(2,3);
}
</style>
</head>
<body>
<h1>The scale() Method</h1>
<p>The scale() method increases or decreases the size of an element.</p>
<div>
```

This div element is two times of its original width, and three times of its original height.

</div>

</body>

</html>

The following example decreases the <div> element to be half of its original width and height:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
```

```

margin: 150px;
width: 200px;
height: 100px;
background-color: yellow;
border: 1px solid black;
transform: scale(0.5,0.5);
}
</style>
</head>
<body>
<h1>The scale() Method</h1>
<p>The scale() method increases or decreases the size of an element.</p>
<div>
This div element is decreased to be half of its original width and height.
</div>
</body>
</html>
```

~ 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:

eg:

```
div {
  transform: scaleX(2);
}
```

~ 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:

eg:

```
div {
  transform: scaleY(3);
}
```

~ The skewX() Method:

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

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

eg:

```
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:

eg:

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

~ The Skew() Method:

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:

eg:

```
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:

eg:

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

~ The Matrix() Method:

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

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

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

eg:

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

CSS Transform Properties

The following table lists all the 2D transform properties:

Property	Description
<code>transform</code>	Applies a 2D or 3D transformation to an element
<code>transform-origin</code>	Allows you to change the position on transformed elements

CSS 2D Transform Methods

Function	Description
<code>matrix(n,n,n,n,n,n)</code>	Defines a 2D transformation, using a matrix of six values
<code>translate(x,y)</code>	Defines a 2D translation, moving the element along the X- and the Y-axis
<code>translateX(n)</code>	Defines a 2D translation, moving the element along the X-axis
<code>translateY(n)</code>	Defines a 2D translation, moving the element along the Y-axis
<code>scale(x,y)</code>	Defines a 2D scale transformation, changing the element's width and height
<code>scaleX(n)</code>	Defines a 2D scale transformation, changing the element's width
<code>scaleY(n)</code>	Defines a 2D scale transformation, changing the element's height
<code>rotate(angle)</code>	Defines a 2D rotation, the angle is specified in the parameter
<code>skew(x-angle,y-angle)</code>	Defines a 2D skew transformation along the X- and the Y-axis
<code>skewX(angle)</code>	Defines a 2D skew transformation along the X-axis
<code>skewY(angle)</code>	Defines a 2D skew transformation along the Y-axis

CSS 3D Transforms

CSS also supports 3D transformations.

- 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:

eg:

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

~ *The rotateY() Method:*

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

eg:

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

~ *The rotateZ() Method:*

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

eg:

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

CSS Transform Properties

The following table lists all the 3D transform properties:

Property	Description
<code>transform</code>	Applies a 2D or 3D transformation to an element
<code>transform-origin</code>	Allows you to change the position on transformed elements
<code>transform-style</code>	Specifies how nested elements are rendered in 3D space
<code>perspective</code>	Specifies the perspective on how 3D elements are viewed
<code>perspective-origin</code>	Specifies the bottom position of 3D elements
<code>backface-visibility</code>	Defines whether or not an element should be visible when not facing the screen

CSS 3D Transform Methods

Function	Description
<code>matrix3d(n,n,n,n,n,n,n,n,n,n,n,n,n,n)</code>	Defines a 3D transformation, using a 4x4 matrix of 16 values
<code>translate3d(x,y,z)</code>	Defines a 3D translation
<code>translateX(x)</code>	Defines a 3D translation, using only the value for the X-axis
<code>translateY(y)</code>	Defines a 3D translation, using only the value for the Y-axis
<code>translateZ(z)</code>	Defines a 3D translation, using only the value for the Z-axis
<code>scale3d(x,y,z)</code>	Defines a 3D scale transformation
<code>scaleX(x)</code>	Defines a 3D scale transformation by giving a value for the X-axis
<code>scaleY(y)</code>	Defines a 3D scale transformation by giving a value for the Y-axis
<code>scaleZ(z)</code>	Defines a 3D scale transformation by giving a value for the Z-axis
<code>rotate3d(x,y,z,angle)</code>	Defines a 3D rotation
<code>rotateX(angle)</code>	Defines a 3D rotation along the X-axis
<code>rotateY(angle)</code>	Defines a 3D rotation along the Y-axis
<code>rotateZ(angle)</code>	Defines a 3D rotation along the Z-axis
<code>perspective(n)</code>	Defines a perspective view for a 3D transformed element

CSS Animations

CSS allows animation of HTML elements without using JavaScript!

What are CSS Animations?

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

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

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

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

- The `@keyframes` Rule:

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

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

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
```

```

width: 100px;
height: 100px;
background-color: red;
animation-name: example;
animation-duration: 4s;
}
@keyframes example {
from {background-color: red;}
to {background-color: yellow;}
}
</style>
</head>
<body>
<h1>CSS Animation</h1>
<div></div>
<p><b>Note:</b> When an animation is finished, it goes back to its original style.</p>
</body>
</html>
# Note: The animation-duration property defines how long an animation should take to complete. If the animation-duration property is not specified, no animation will occur, because the default value is 0s (0 seconds).
In the example above we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)).
It is also possible to use percent. By using percent, you can add as many style changes as you like.
~ The following example will change the background-color of the <div> element when the animation is 25% complete, 50% complete, and again when the animation is 100% complete:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
div {
width: 100px;
height: 100px;
background-color: red;
animation-name: example;
animation-duration: 4s;
}
@keyframes example {
0% {background-color: red;}
25% {background-color: yellow;}
50% {background-color: blue;}
100% {background-color: green;}
}
</style>
</head>

```

```

<body>
<h1>CSS Animation</h1>
<div></div>
<p><b>Note:</b> When an animation is finished, it goes back to its original style.</p>
</body>
</html>
~ 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:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  position: relative;
  animation-name: example;
  animation-duration: 4s;
}
@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;}
}
</style>
</head>
<body>
<h1>CSS Animation</h1>
<div></div>
<p><b>Note:</b> When an animation is finished, it goes back to its original style.</p>
</body>
</html>
• 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:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
}

```

```

background-color: red;
position: relative;
animation-name: example;
animation-duration: 4s;
animation-delay: 2s;
}
@keyframes example {
0% {background-color:red; left:0px; top:0px;}
25% {background-color:yellow; left:200px; top:0px;}
50% {background-color:blue; left:200px; top:200px;}
75% {background-color:green; left:0px; top:200px;}
100% {background-color:red; left:0px; top:0px;}
}
</style>
</head>
<body>
<h1>CSS Animation</h1>
<p>The animation-delay property specifies a delay for the start of an animation. The following example has a 2 seconds delay before starting the animation:</p>
<div></div>
</body>
</html>
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:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
div {
width: 100px;
height: 100px;
background-color: red;
position: relative;
animation-name: example;
animation-duration: 4s;
animation-delay: -2s;
}
@keyframes example {
0% {background-color:red; left:0px; top:0px;}
25% {background-color:yellow; left:200px; top:0px;}
50% {background-color:blue; left:200px; top:200px;}
75% {background-color:green; left:0px; top:200px;}
100% {background-color:red; left:0px; top:0px;}
}
</style>

```

```

</head>
<body>
<h1>CSS Animation</h1>
<p>Using negative values in the animation-delay property: Here, the animation will start as if it had already been playing for 2 seconds:</p>
<div></div>
</body>
</html>

```

- 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:*

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background-color: red;
  position: relative;
  animation-name: example;
  animation-duration: 4s;
  animation-iteration-count: 3;
}

```

```

@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;}
}

```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>CSS Animation</h1>
```

<p>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:</p>

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

```
</body>
```

```
</html>
```

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

eg:

```

div {
  width: 100px;
  height: 100px;
  position: relative;
}
```

```

background-color: red;
animation-name: example;
animation-duration: 4s;
animation-iteration-count: infinite;
}

```

- Run Animation in Reverse Direction or Alternate Cycles:

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

The animation-direction property can have the following values:

normal - The animation is played as normal (forwards). This is default

reverse - The animation is played in reverse direction (backwards)

alternate - The animation is played forwards first, then backwards

alternate-reverse - The animation is played backwards first, then forwards

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

eg:

```

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

```

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

eg:

```

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

```

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

eg:

```

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

```

```
animation-direction: alternate-reverse;
}
```

- Specify the Speed Curve of the Animation:

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

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

ease - Specifies an animation with a slow start, then fast, then end slowly (this is default)

linear - Specifies an animation with the same speed from start to end

ease-in - Specifies an animation with a slow start

ease-out - Specifies an animation with a slow end

ease-in-out - Specifies an animation with a slow start and end

cubic-bezier(n,n,n,n) - Lets you define your own values in a cubic-bezier function

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

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 50px;
  background-color: red;
  font-weight: bold;
  position: relative;
  animation: mymove 5s;
  animation-fill-mode: forwards;
}
#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;}
@keyframes mymove {
  from {left: 0px;}
  to {left: 300px;}
}
</style>
</head>
<body>
<h1>CSS Animation</h1>
<p>The animation-timing-function property specifies the speed curve of the animation. The following example shows some of the different speed curves that can be used:</p>
<div id="div1">linear</div>
<div id="div2">ease</div>
<div id="div3">ease-in</div>
<div id="div4">ease-out</div>
<div id="div5">ease-in-out</div>
</body>
</html>
```

- 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

eg:

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

~ The following example lets the <div> element get the style values set by the first keyframe before the animation starts (during the animation-delay period):

eg:

```
div {
    width: 100px;
    height: 100px;
    background: red;
    position: relative;
    animation-name: example;
    animation-duration: 3s;
    animation-delay: 2s;
    animation-fill-mode: backwards;
}
```

~ The following example lets the <div> element get the style values set by the first keyframe before the animation starts, and retain the style values from the last keyframe when the animation ends:

eg:

```
div {
    animation-name: example;
    animation-duration: 5s;
    animation-timing-function: linear;
    animation-delay: 2s;
```

```

animation-iteration-count: infinite;
animation-direction: alternate;
}

```

- Animation Shorthand Property:

~ *The example below uses six of the animation properties:*

eg:

```

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:*

eg:

```

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

```

CSS Tooltip

A tooltip is often used to specify extra information about something when the user moves the mouse pointer over an element:

- Basic Tooltip:

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

eg:

```

<style>
/* Tooltip container */
.tooltip {
    position: relative;
    display: inline-block;
    border-bottom: 1px dotted black; /* If you want dots under the hoverable text */
}
/* Tooltip text */
.tooltip .tooltiptext {
    visibility: hidden;
    width: 120px;
    background-color: black;
    color: #fff;
    text-align: center;
    padding: 5px 0;
    border-radius: 6px;
    /* Position the tooltip text - see examples below! */
    position: absolute;
    z-index: 1;
}

```

```

}
/* Show the tooltip text when you mouse over the tooltip container */
.tooltip:hover .tooltiptext {
  visibility: visible;
}
</style>
<div class="tooltip">Hover over me
  <span class="tooltiptext">Tooltip text</span>
</div>

```

Example Explained

HTML: Use a container element (like `<div>`) and add the "tooltip" class to it. When the user mouse over this `<div>`, it will show the tooltip text.

The tooltip text is placed inside an inline element (like ``) with `class="tooltiptext"`.

CSS: The tooltip class use `position:relative`, which is needed to position the tooltip text (`position:absolute`). Note: See examples below on how to position the tooltip.

The `.tooltiptext` class holds the actual tooltip text. It is hidden by default, and will be visible on hover (see below). We have also added some basic styles to it: 120px width, black background color, white text color, centered text, and 5px top and bottom padding.

The CSS `border-radius` property is used to add rounded corners to the tooltip text.

The `:hover` selector is used to show the tooltip text when the user moves the mouse over the `<div>` with `class="tooltip"`.

• 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.

eg:

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

~ Left Tooltip:

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

If you want the tooltip to appear on top or on the bottom, see examples below. Note that we use the `margin-left` property with a value of minus 60 pixels. This is to center the tooltip above/below the hoverable text. It is set to the half of the tooltip's width ($120/2 = 60$)

~ Top Tooltip:

eg:

```
.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:

~ Bottom Arrow:

eg:

```
.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;
}
```

Example Explained

Position the arrow inside the tooltip: top: 100% will place the arrow at the bottom of the tooltip. left: 50% will center the arrow.

Note: The border-width property specifies the size of the arrow. If you change this, also change the margin-left value to the same. This will keep the arrow centered.

The border-color is used to transform the content into an arrow. We set the top border to black, and the rest to transparent. If all sides were black, you would end up with a black square box.

This example demonstrates how to add an arrow to the top of the tooltip. Notice that we set the bottom border color this time:

~ Top Arrow:

eg:

```
.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;
}
```

~ This example demonstrates how to add an arrow to the left of the tooltip:

eg:

```
.tooltip .tooltiptext::after {
  content: " ";
  position: absolute;
  top: 50%;
  right: 100%; /* To the left of the tooltip */
  margin-top: -5px;
  border-width: 5px;
  border-style: solid;
  border-color: transparent black transparent transparent;
}

~ This example demonstrates how to add an arrow to the right of the tooltip:

.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):

eg:

```
.tooltip .tooltiptext {
  opacity: 0;
  transition: opacity 1s;
}

.tooltip:hover .tooltiptext {
  opacity: 1;
}
```

CSS Styling Images

- Rounded Images:

Use the border-radius property to create rounded images:

eg 1: Rounded Image

```
img {
  border-radius: 8px;
}
```

eg 2: Circled Image

```
img {
  border-radius: 50%;
}
```

- Thumbnail Images:

Use the border property to create thumbnail images.

eg 1:Thumbnail Image

```
img {
    border: 1px solid #ddd;
    border-radius: 4px;
    padding: 5px;
    width: 150px;
}

```

eg 2: Thumbnail Image as a link

```
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);
}
<a href="paris.jpg">
    
</a>
```

- Responsive Images:

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

If you want an image to scale down if it has to, but never scale up to be larger than its original size, add the following:

eg:

```
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:

eg:

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

- Polaroid Images / Cards:

eg:

```
div.polaroid {
    width: 80%;
    background-color: white;
    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
}
img {width: 100%}
```

```
div.container {
    text-align: center;
    padding: 10px 20px;
}
```

- Transparent Image:

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

eg:

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

- Image Text:

~ Top Left:

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
    position: relative;
}
.topleft {
    position: absolute;
    top: 8px;
    left: 16px;
    font-size: 18px;
}
img {
    width: 100%;
    height: auto;
    opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the top left corner:</p>
<div class="container">
    
    <div class="topleft">Top Left</div>
</div>
</body>
</html>
```

~ Top Right:

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

```
}

.topright {
    position: absolute;
    top: 8px;
    right: 16px;
    font-size: 18px;
}
img {
    width: 100%;
    height: auto;
    opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the top right corner:</p>
<div class="container">
    
    <div class="topright">Top Right</div>
</div>
</body>
</html>
~ Bottom Left:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
    position: relative;
}
.bottomleft {
    position: absolute;
    bottom: 8px;
    left: 16px;
    font-size: 18px;
}
img {
    width: 100%;
    height: auto;
    opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the bottom left corner:</p>
<div class="container">
```

```

<div class="bottomleft">Bottom Left</div>
</div>
</body>
</html>
~ Bottom Right:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.bottomright {
  position: absolute;
  bottom: 8px;
  right: 16px;
  font-size: 18px;
}
img {
  width: 100%;
  height: auto;
  opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Add some text to an image in the bottom right corner:</p>
<div class="container">
  
  <div class="bottomright">Bottom Right</div>
</div>
</body>
</html>
~ Centered:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
}
.center {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

```

    font-size: 18px;
}
img {
    width: 100%;
    height: auto;
    opacity: 0.3;
}
</style>
</head>
<body>
<h2>Image Text</h2>
<p>Center text in image:</p>
<div class="container">
    
    <div class="center">Centered</div>
</div>
</body>
</html>

```

• Image Filters:

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

Note: The filter property is not supported in Internet Explorer or Edge 12

eg:

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

```
img {
    filter: grayscale(100%);
}
```

~ All Filters example:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color:white;
}
img {
    width: 33%;
    height: auto;
    float: left;
    max-width: 235px;
}
.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);}
</style>
</head>
<body>
<h2>Image Filters</h2>
<p><strong>Note:</strong> The filter property is not supported in Internet Explorer or Edge 12.</p>











</body>
</html>

```

• Image Hover Overlay:

~ Fade in Text:

```

<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
  width: 50%;
}
.image {
  display: block;
  width: 100%;
  height: auto;
}
.overlay {
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  height: 100%;
  width: 100%;
  opacity: 0;
  transition: .5s ease;
  background-color: #008CBA;
}

```

```
.container:hover .overlay {
  opacity: 1;
}
.text {
  color: white;
  font-size: 20px;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  -ms-transform: translate(-50%, -50%);
}
</style>
</head>
<body>
<h2>Fade in Overlay</h2>
<div class="container">
  
  <div class="overlay">
    <div class="text">Hello World</div>
  </div>
</div>
</body>
</html>
~ Fade in Box:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
  width: 50%;
}
.image {
  opacity: 1;
  display: block;
  width: 100%;
  height: auto;
  transition: .5s ease;
  backface-visibility: hidden;
}
.middle {
  transition: .5s ease;
  opacity: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
```

```
-ms-transform: translate(-50%, -50%)
}
.container:hover .image {
  opacity: 0.3;
}
.container:hover .middle {
  opacity: 1;
}
.text {
  background-color: #4CAF50;
  color: white;
  font-size: 16px;
  padding: 16px 32px;
}

```

</style>

```
</head>
<body>
<h2>Fade in a Box</h2>
<div class="container">
  
  <div class="middle">
    <div class="text">John Doe</div>
  </div>
</div>
</body>
</html>
```

~ Slide in Top:

```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
  width: 50%;
}
.image {
  display: block;
  width: 100%;
  height: auto;
}
.overlay {
  position: absolute;
  bottom: 100%;
  left: 0;
  right: 0;
  background-color: #008CBA;
  overflow: hidden;
  width: 100%;
```

```
height: 0;
transition: .5s ease;
}
.container:hover .overlay {
bottom: 0;
height: 100%;
}
.text {
white-space: nowrap;
color: white;
font-size: 20px;
position: absolute;
overflow: hidden;
top: 50%;
left: 50%;
transform: translate(-50%, -50%);
-ms-transform: translate(-50%, -50%);
}
</style>
</head>
<body>
<h2>Slide in Overlay from the Top</h2>
<div class="container">

<div class="overlay">
<div class="text">Hello World</div>
</div>
</div>
</body>
</html>
~ Slide in Bottom:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
position: relative;
width: 50%;
}
.image {
display: block;
width: 100%;
height: auto;
}
.overlay {
position: absolute;
bottom: 0;
left: 0;
```

```

right: 0;
background-color: #008CBA;
overflow: hidden;
width: 100%;
height: 0;
transition: .5s ease;
}
.container:hover .overlay {
  height: 100%;
}
.text {
  white-space: nowrap;
  color: white;
  font-size: 20px;
  position: absolute;
  overflow: hidden;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  -ms-transform: translate(-50%, -50%);
}

```

</style>

</head>

<body>

<h2>Slide in Overlay from the Bottom</h2>

<div class="container">

 <div class="overlay">

 <div class="text">Hello World</div>

 </div>

</div>

</body>

</html>

~ Slide in Left:

<!DOCTYPE html>

<html>

<head>

<style>

```

.container {
  position: relative;
  width: 50%;
}

```

.image {

display: block;
 width: 100%;
 height: auto;
}

.overlay {

```
position: absolute;
bottom: 0;
left: 0;
right: 0;
background-color: #008CBA;
overflow: hidden;
width: 0;
height: 100%;
transition: .5s ease;
}
.container:hover .overlay {
  width: 100%;
}
.text {
  white-space: nowrap;
  color: white;
  font-size: 20px;
  position: absolute;
  overflow: hidden;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  -ms-transform: translate(-50%, -50%);
}
</style>
</head>
<body>
<h2>Slide in Overlay from the Left</h2>
<div class="container">
  
  <div class="overlay">
    <div class="text">Hello World</div>
  </div>
</div>
</body>
</html>
~ Slide in Right:
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  position: relative;
  width: 50%;
}
.image {
  display: block;
  width: 100%;
```

```

height: auto;
}
.overlay {
position: absolute;
bottom: 0;
left: 100%;
right: 0;
background-color: #008CBA;
overflow: hidden;
width: 0;
height: 100%;
transition: .5s ease;
}
.container:hover .overlay {
width: 100%;
left: 0;
}
.text {
white-space: nowrap;
color: white;
font-size: 20px;
position: absolute;
overflow: hidden;
top: 50%;
left: 50%;
transform: translate(-50%, -50%);
-ms-transform: translate(-50%, -50%);
}

```

</style>

</head>

<body>

<h2>Slide in Overlay from the Right</h2>

<div class="container">

<div class="overlay">

<div class="text">Hello World</div>

</div>

</div>

</body>

</html>

- [Flip an Image:](#)

eg:

```

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

```

- [Responsive Image Gallery:](#)

```

.responsive {
padding: 0 6px;

```

```

float: left;
width: 24.99999%;
}
@media only screen and (max-width: 700px){
.responsive {
width: 49.99999%;
margin: 6px 0;
}
}
@media only screen and (max-width: 500px){
.responsive {
width: 100%;
}
}

```

- [Image Modal \(Advanced\)](#):

This is an example to demonstrate how CSS and JavaScript can work together.

First, use CSS to create a modal window (dialog box), and hide it by default.

Then, use a JavaScript to show the modal window and to display the image inside the modal, when a user clicks on the image:

eg:

```

// Get the modal
var modal = document.getElementById('myModal');
// Get the image and insert it inside the modal - use its "alt" text as a caption
var img = document.getElementById('myImg');
var modalImg = document.getElementById("img01");
var captionText = document.getElementById("caption");
img.onclick = function(){
modal.style.display = "block";
modalImg.src = this.src;
captionText.innerHTML = this.alt;
}
// Get the <span> element that closes the modal
var span = document.getElementsByClassName("close")[0];
// When the user clicks on <span> (x), close the modal
span.onclick = function() {
modal.style.display = "none";
}

```

CSS Image Reflection

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.

eg:

~ Here we want the reflection below the image:

```

img {
-webkit-box-reflect: below;
}
```

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

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

- Reflection Offset:

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

eg:

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

- Reflection With Gradient:

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

eg:

Create a fade-out effect on the reflection:

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

CSS Object-Fit Property

The CSS object-fit property is used to specify how an 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".

If we won't use the Object fit property the image will be squished to fit the container.

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 squashed 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.

eg:

```
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.

eg:

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

- 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.

eg:

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

- Using object-fit: none; :

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

eg:

```
img {
  width: 200px;
  height: 300px;
  object-fit: none;
}
```

- 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.

eg:

```
img {
  width: 200px;
  height: 300px;
  object-fit: scale-down;
}
```

~ *The following example demonstrates all the possible values of the object-fit property in one example:*

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.fill {object-fit: fill;}
.contain {object-fit: contain;}
.cover {object-fit: cover;}
.scale-down {object-fit: scale-down;}
.none {object-fit: none;}
</style>
</head>
<body>
<h1>The object-fit Property</h1>
<h2>No object-fit:</h2>
```

```


<h2>object-fit: fill (this is default):</h2>

<h2>object-fit: contain:</h2>

<h2>object-fit: cover:</h2>

<h2>object-fit: scale-down:</h2>

<h2>object-fit: none:</h2>

</body>
</html>

```

CSS The object-position property

The CSS object-position property is used to specify how an `` or `<video>` should be positioned within its container.

From This image reference:



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.

Here we will use the object-position property to position the image so that the great old building is in center:



eg:

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

Here we will use the object-position property to position the image so that the famous Eiffel Tower is in center:



eg:

```
img {
  width: 200px;
  height: 300px;
  object-fit: cover;
  object-position: 15% 100%;
}
```

CSS Masking

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

- 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, or an SVG <mask> 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.



Then this will be the output:



eg:

Source Code:

```
<!DOCTYPE html>
<html>
<head>
<style>
.mask1 {
-webkit-mask-image: url(w3logo.png);
mask-image: url(w3logo.png);
-webkit-mask-repeat: no-repeat;
mask-repeat: no-repeat;
}
</style>
</head>
<body>
<h1>The mask-image Property</h1>
<h3>An image with a mask layer image:</h3>
<div class="mask1">

</div>
<h3>Original image:</h3>

```

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

Example Explained

The mask-image property specifies the image to be used as a mask layer for an element. The mask-repeat property specifies if or how a mask image will be repeated. The no-repeat value indicates that the mask image will not be repeated (the mask image will only be shown once).

If we won't use the no-repeat then this will be the output.



Source Code:

```
.mask1 {
  -webkit-mask-image: url(w3logo.png);
  mask-image: url(w3logo.png);
}
```

- Use Gradients as the Mask Layer:

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

~~ Linear Gradient Example:

Here, we use a linear-gradient as the mask layer for our image. This linear gradient goes from top (black) to bottom (transparent):



eg:

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

~ Use a linear gradient along with text masking as a mask layer

eg:

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

```

<head>
<style>
p {
  font-size: 20px;
  padding: 20px;
  color: white;
}
.mask1 {
  max-width: 600px;
  height: 350px;
  overflow-y: scroll;
  background: url(img_5terre.jpg) no-repeat;
  -webkit-mask-image: linear-gradient(black, transparent);
  mask-image: linear-gradient (black, transparent);
}
</style>
</head>
<body>
<h1>The mask-image Property</h1>
<h3>A linear gradient as a mask layer:</h3>
<div class="mask1">
<p>The Cinque Terre is a coastal area within Liguria, in the northwest of Italy. It lies in the west of La Spezia Province, and comprises five villages: Monterosso al Mare, Vernazza, Corniglia, Manarola, and Riomaggiore.</p>
<p>The Cinque Terre is a coastal area within Liguria, in the northwest of Italy. It lies in the west of La Spezia Province, and comprises five villages: Monterosso al Mare, Vernazza, Corniglia, Manarola, and Riomaggiore.</p>
<p>The Cinque Terre is a coastal area within Liguria, in the northwest of Italy. It lies in the west of La Spezia Province, and comprises five villages: Monterosso al Mare, Vernazza, Corniglia, Manarola, and Riomaggiore.</p>
</div>
</body>
</html>
~~ Radial Gradient Example:

```

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



eg:

```
.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%);
```

```

}
(Shaped as 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%);
}

```

- Use SVG as the Mask Layer:

The SVG <mask> element can be used inside an SVG graphic to create masking effects.

Here, we use the SVG <mask> element to create different mask layers for our image:



eg:

~ An SVG mask layer (Formed as a Triangle).

```

<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1>The mask-image Property</h1>
<h3>An SVG mask layer (formed as a triangle):</h3>
<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>
<h3>Original image:</h3>

</body>
</html>

```

~ 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 a Circle).

```
<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>
```

- CSS Masking Properties:

Property	Description
<u>mask-image</u>	Specifies an image to be used as a mask layer for an element
<u>mask-mode</u>	Specifies whether the mask layer image is treated as a luminance mask or as an alpha mask
<u>mask-origin</u>	Specifies the origin position (the mask position area) of a mask layer image
<u>mask-position</u>	Sets the starting position of a mask layer image (relative to the mask position area)
<u>mask-repeat</u>	Specifies how the mask layer image is repeated
<u>mask-size</u>	Specifies the size of a mask layer image

CSS Buttons

Example Button:

```
<!DOCTYPE html>
<html>
<head>
<style>
.button {
  background-color: #04AA6D;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
}
</style>
```

```

</head>
<body>
<h2>CSS Buttons</h2>
<button>Default Button</button>
<a href="#" class="button">Link Button</a>
<button class="button">Button</button>
<input type="button" class="button" value="Input Button">
</body>
</html>

```

- Button Colors:

Use the background-color property to change the background color of a button:

eg:

```

.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:

Use the font-size property to change the font size of a button:

eg:

```

.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:

eg:

```

.button1 {padding: 10px 24px;}
.button2 {padding: 12px 28px;}
.button3 {padding: 14px 40px;}
.button4 {padding: 32px 16px;}
.button5 {padding: 16px;}

```

- Rounded Buttons:

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

```

.button1 {border-radius: 2px;}
.button2 {border-radius: 4px;}
.button3 {border-radius: 8px;}
.button4 {border-radius: 12px;}
.button5 {border-radius: 50%;}

```

- Coloured Button Borders:

Use the border property to add a colored border to a button:

eg:

```

.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:

eg:

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

- [Shadow Buttons](#):

Use the box-shadow property to add shadows to a button:

eg:

```
.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:

eg:

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

- [Button Width](#):

By default, the size of the button is determined by its text content (as wide as its content).

Use the width property to change the width of a button:

eg:

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

- [Button Groups](#):

Remove margins and add float:left to each button to create a button group.

eg:

```
.button {  
    float: left;  
}
```

- [Bordered Button Group](#):

Use the border property to create a bordered button group:

eg:

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

- Vertical Button Group:

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

eg:

```
.button {
  display: block;
}
```

- Button on Image:

eg:

```
<!DOCTYPE html>
<html>
<head>
<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">
  
  <button class="btn">Buy Now</button>
</div>
</body>
</html>
```

```
<button class="btn">Button</button>
</div>
</body>
</html>
• Animated Button:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
.button {
    display: inline-block;
    border-radius: 4px;
    background-color: #f4511e;
    border: none;
    color: #FFFFFF;
    text-align: center;
    font-size: 28px;
    padding: 20px;
    width: 200px;
    transition: all 0.5s;
    cursor: pointer;
    margin: 5px;
}
.button span {
    cursor: pointer;
    display: inline-block;
    position: relative;
    transition: 0.5s;
}
.button span:after {
    content: '\00bb';
    position: absolute;
    opacity: 0;
    top: 0;
    right: -20px;
    transition: 0.5s;
}
.button:hover span {
    padding-right: 25px;
}
.button:hover span:after {
    opacity: 1;
    right: 0;
}
</style>
</head>
<body>
```

```

<h2>Animated Button</h2>
<button class="button" style="vertical-align:middle"><span>Hover </span></button>
</body>
</html>
• Pressed Effect on Button:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
.button {
    display: inline-block;
    padding: 15px 25px;
    font-size: 24px;
    cursor: pointer;
    text-align: center;
    text-decoration: none;
    outline: none;
    color: #fff;
    background-color: #04AA6D;
    border: none;
    border-radius: 15px;
    box-shadow: 0 9px #999;
}
.button:hover {background-color: #3e8e41}
.button:active {
    background-color: #3e8e41;
    box-shadow: 0 5px #666;
    transform: translateY(4px);
}
</style>
</head>
<body>
<h2>Animated Button - "Pressed Effect"</h2>
<button class="button">Click Me</button>
</body>
</html>
• Fade in on Hover:
eg:
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
.button {
    background-color: #f4511e;
    border: none;
    color: white;

```

```

padding: 16px 32px;
text-align: center;
font-size: 16px;
margin: 4px 2px;
opacity: 0.6;
transition: 0.3s;
display: inline-block;
text-decoration: none;
cursor: pointer;
}
.button:hover {opacity: 1}
</style>
</head>
<body>
<h2>Animated Button - Fade in Effect</h2>
<button class="button">Hover Over Me</button>
</body>
</html>
• Ripple Effect on Click:
eg:
<!DOCTYPE html>
<html>
<head>
<style>
.button {
position: relative;
background-color: #04AA6D;
border: none;
font-size: 28px;
color: #FFFFFF;
padding: 20px;
width: 200px;
text-align: center;
transition-duration: 0.4s;
text-decoration: none;
overflow: hidden;
cursor: pointer;
}
.button:after {
content: "";
background: #f1f1f1;
display: block;
position: absolute;
padding-top: 300%;
padding-left: 350%;
margin-left: -20px !important;
margin-top: -120%;
opacity: 0;
}

```

```

transition: all 0.8s
}
.button:active:after {
  padding: 0;
  margin: 0;
  opacity: 1;
  transition: 0s
}
</style>
</head>
<body>
<h2>Animated Button - Ripple Effect</h2>
<button class="button">Click Me</button>
</body>
</html>

```

CSS Pagination

- [Simple Pagination:](#)

```

<!DOCTYPE html>
<html>
<head>
<style>
.pagination {
  display: inline-block;
}
.pagination a {
  color: black;
  float: left;
  padding: 8px 16px;
  text-decoration: none;
}
</style>
</head>
<body>
<h2>Simple Pagination</h2>
<div class="pagination">
  <a href="#">&laquo;</a>
  <a href="#">1</a>
  <a href="#">2</a>
  <a href="#">3</a>
  <a href="#">4</a>
  <a href="#">5</a>
  <a href="#">6</a>
  <a href="#">&raquo;</a>
</div>
</body>

```

</html>

- 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

eg:

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

- Rounded Active and Hoverable Buttons:

Add the border-radius property if you want a rounded "active" and "hover" button.

eg:

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

- Hoverable Transition Effect:

Add the transition property to the page links to create a transition effect on hover.

eg:

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

- Bordered Pagination:

Use the border property to add borders to the pagination:

eg:

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

- Rounded Borders:

```
.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:

eg:

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

- Pagination Size:

Change the size of the pagination with the font-size property:

eg:

```
.pagination a {
  font-size: 22px;
}
```

- Centered Pagination:

To center the pagination, wrap a container element (like <div>) around it with text-align:center.

eg:

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

- More Examples:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color:white;
}
.pagination {
  display: inline-block;
}
.pagination a {
  color: black;
  float: left;
  padding: 8px 16px;
  text-decoration: none;
  transition: background-color .3s;
  border: 1px solid #ddd;
}
.pagination a.active {
  background-color: #4CAF50;
  color: white;
  border: 1px solid #4CAF50;
}
.pagination a:hover:not(.active) {background-color: #ddd;}
</style>
</head>
<body>
<p>Next/Previous buttons:</p>
<div class="pagination">
  <a href="#"><</a>
  <a href="#">></a>
</div>
<p>Navigation pagination:</p>
<div class="pagination">
  <a href="#" class="active">Home</a>
  <a href="#">Link 1</a>
```

```

<a href="#">Link 2</a>
<a href="#">Link 3</a>
</div>
</body>
</html>

```

- Bread Crumbs:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
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";
}
ul.breadcrumb li a {color: green;}
</style>
</head>
<body>
<h2>Breadcrumb Pagination</h2>
<ul class="breadcrumb">
  <li><a href="#">Home</a></li>
  <li><a href="#">Pictures</a></li>
  <li><a href="#">Summer 15</a></li>
  <li>Italy</li>
</ul>
</body>
</html>

```

CSS Multiple Columns

- CSS Multi-column Layout:

The CSS multi-column layout allows easy definition of multiple columns of text - just like in newspapers:



- Create Multiple Columns:

The column-count property specifies the number of columns an element should be divided into.

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.newspaper {
  column-count: 3;
}
</style>
</head>
<body>
<h1>Create Multiple Columns</h1>
<div class="newspaper">
  Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.
</div>
</body>
</html>
```

- Specify The Gap Between Columns:

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

eg:

```
div {
  column-gap: 40px;
}
```

- Column Rules:

~ The **column-rule-style** property specifies the style of the rule between columns.

eg:

```
div {
  column-rule-style: solid;
}
```

~ The **column-rule-width** property specifies the width of the rule between columns.

eg:

```
div {
  column-rule-width: 1px;
}
```

~ The **column-rule-color** property specifies the color of the rule between columns.

eg:

```
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

eg:

```
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:

eg:

```
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:

eg:

```
div {
  column-width: 100px;
}
```

CSS User Interface

- CSS Resizing:

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

~ The following example lets the user resize only the width of a <div> element:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    border: 2px solid;
    padding: 20px;
    width: 300px;
    resize: horizontal;
    overflow: auto;
}
</style>
</head>
<body>
<h1>The resize Property</h1>
<div>
    <p>Let the user resize only the width of this div element.</p>
    <p>To resize: Click and drag the bottom right corner of this div element.</p>
</div>
</body>
</html>
```

~ The following example lets the user resize only the height of a <div> element:

eg:

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

~ The following example lets the user resize both the height and width of a <div> element:

eg:

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

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

eg:

```
textarea {
    resize: none;
}
```

- 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.
The following example uses the outline-offset property to add space between the border and the outline:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
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;
}
</style>
</head>
<body>
<h1>The outline-offset Property</h1>
<div class="ex1">This div has a 4 pixels solid red outline 15 pixels outside the border edge.</div>
<br>
<div class="ex2">This div has a 5 pixels dashed blue outline 5 pixels outside the border edge.</div>
</body>
</html>
```

CSS Variables - The var() Function

- 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 copying and pasting the same colors over and over again, you can place them in variables.

- The Traditional Way:

The following example shows the traditional way of defining some colors in a style sheet (by defining the colors to use, for each specific element):

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: #1e90ff;
}
h2 {
  border-bottom: 2px solid #1e90ff;
}
.container {
  color: #1e90ff;
  background-color: #ffffff;
  padding: 15px;
}
button {
  background-color: #ffffff;
  color: #1e90ff;
  border: 1px solid #1e90ff;
  padding: 5px;
}
</style>
</head>
<body>
<h1>The Traditional Way</h1>
<div class="container">
  <h2>Lorem Ipsum</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam semper diam at erat
  pulvinar, at pulvinar felis blandit.</p>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam semper diam at erat
  pulvinar, at pulvinar felis blandit.</p>
  <p>
    <button>Yes</button>
    <button>No</button>
  </p>
</div>
</body>
</html>

```

- 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 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.

The following example is equal to the example above, but here we use the var() function.

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:

eg:

```
: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

To change the blue and white color to a softer blue and white, you just need to change the two variable values:

eg:

```
:root {
  --blue: #6495ed;
  --white: #faf0e6;
}

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;}
```

CSS Variables

- CSS variables:

The var () Function

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 copying and pasting the same colors over and over again, you can place them in variables.

The Traditional Way:

The following example shows the traditional way of defining some colors in a style sheet (by defining the colors to use, for each specific element):

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: #1e90ff;
}
h2 {
    border-bottom: 2px solid #1e90ff;
}
.container {
    color: #1e90ff;
    background-color: #ffffff;
    padding: 15px;
}
button {
    background-color: #ffffff;
    color: #1e90ff;
    border: 1px solid #1e90ff;
    padding: 5px;
}
</style>
</head>
<body>
<h1>The Traditional Way</h1>
<div class="container">
    <h2>Lorem Ipsum</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam semper diam at erat pulvinar, at pulvinar felis blandit.</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam semper diam at erat pulvinar, at pulvinar felis blandit.</p>
    <p>
        <button>Yes</button>
        <button>No</button>
    </p>

```

```
</div>
```

```
</body>
```

```
</html>
```

- 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 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.

The following example is equal to the example above, but here we use the var() function.

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:

eg:

```
: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

To change the blue and white color to a softer blue and white, you just need to change the two variable values:

eg:

```
:root {
  --blue: #6495ed;
  --white: #faf0e6;
```

```

}
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;
}

```

- CSS Overriding Variables:

~ ~ 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.

Look at the example from the previous topic:

eg:

```

: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;
}

```

Sometimes we want the variables to change only in a specific section of the page.

Assume we want a different color of blue for button elements. Then, we can re-declare the --blue variable inside the button selector. When we use var(--blue) inside this selector, it will use the local --blue variable value declared here.

We see that the local --blue variable will override the global --blue variable for the button elements::

- Add a New Local Variable:

If a variable is to be used at only one single place, we could also have declared a new local variable, like this:

```
: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 {
  --button-blue: #0000ff; /* new local variable */
  background-color: var(--white);
  color: var(--button-blue);
  border: 1px solid var(--button-blue);
  padding: 5px;
}
```

- Change Variables With JavaScript:

CSS variables have access to the DOM, which means that you can change them with JavaScript.

Here is an example of how you can create a script to display and change the --blue variable from the example used in the previous examples:

eg:

```
<script>

// Get the root element
var r = document.querySelector(':root');

// Create a function for getting a variable value
function myFunction_get() {
  // Get the styles (properties and values) for the root
  var rs = getComputedStyle(r);
  // Alert the value of the --blue variable
  alert("The value of --blue is: " + rs.getPropertyValue('--blue'));
}

// Create a function for setting a variable value
function myFunction_set() {
  // Set the value of variable --blue to another value (in this case "lightblue")
  r.style.setProperty('--blue', 'lightblue');
}

</script>
```

- Using Variables in Media Queries:

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

(Media Quires means : Media Queries are about defining different style rules for different devices (screens, tablets, mobile phones, etc.).)

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.

eg:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
/* 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;
}
}
</style>
</head>
<body>
<h1>Using Variables in Media Queries</h1>
<div class="container">
<h2>Lorem Ipsum</h2>
```

<p>When the browser's width is less than 450px, the font-size of this div is 25px. When it is 450px or wider, set the --fontsize variable value to 50px. Resize the browser window to see the effect.</p>

</div>

</body>

</html>

~ Here is another example where we also change the value of the --blue variable in the @media rule:

eg:

/* 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;

}

:root {

--blue: lightblue;

}

}

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:

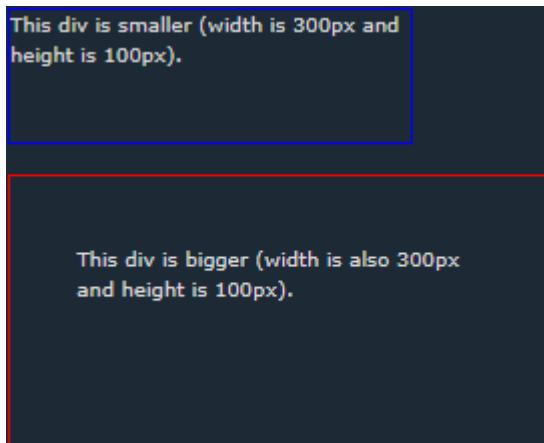
By default, the width and height of an element is calculated like this:

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:



eg:

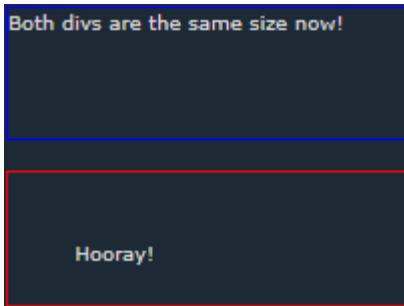
```
<!DOCTYPE html>
<html>
<head>
<style>
.div1 {
  width: 300px;
  height: 100px;
  border: 1px solid blue;
}
.div2 {
  width: 300px;
  height: 100px;
  padding: 50px;
  border: 1px solid red;
}
</style>
</head>
<body>
<h1>Without box-sizing</h1>
<div class="div1">This div is smaller (width is 300px and height is 100px).</div>
<br>
<div class="div2">This div is bigger (width is also 300px and height is 100px).</div>
</body>
</html>
```

The box-sizing property solves this problem.

- With the CSS box-sizing Property:

The box-sizing property allows us to include the padding and border in an element's total width and height.

If you set box-sizing: border-box; on an element, padding and border are included in the width and height:



Here is the same example as above, with box-sizing: border-box; added to both <div> elements:

eg:

```
.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:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    margin: 0;
}
* {
    box-sizing: border-box;
}
input, textarea {
    width: 100%;
}
</style>
```

```

</head>
<body>
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br>
  Comments:<br>
  <textarea name="message" rows="5" cols="30">
  </textarea>
  <br><br>
  <input type="submit" value="Submit">
</form>
<p><strong>Tip:</strong> Try to remove the box-sizing property from the style element and look what happens.
Notice that the width of input, textarea, and submit button will go outside of the screen.</p>
</body>
</html>

```

CSS 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 stylesheet 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):

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: pink;
}
@media screen and (min-width: 480px) {
  body {
    background-color: lightgreen;
  }
}
</style>
</head>
<body>
<h1>Resize the browser window to see the effect!</h1>
```

<p>The media query will only apply if the media type is screen and the viewport is 480px wide or wider.</p>

```

</body>
</html>

~ The following example shows a menu that will float to the left of the page if the viewport is 480 pixels wide or wider (if the viewport is less than 480 pixels, the menu will be on top of the content):
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
.wrapper {overflow: auto;}
#main {margin-left: 4px;}
#leftsidebar {
  float: none;
  width: auto;
}
#menulist {
  margin: 0;
  padding: 0;
}
.menuitem {
  background: #CDF0F6;
  border: 1px solid #d4d4d4;
  border-radius: 4px;
  list-style-type: none;
  margin: 4px;
  padding: 2px;
}
@media screen and (min-width: 480px) {
  #leftsidebar {width: 200px; float: left;}
  #main {margin-left: 216px;}
}
</style>
</head>
<body>
<div class="wrapper">
  <div id="leftsidebar">
    <ul id="menulist">
      <li class="menuitem">Menu-item 1</li>
      <li class="menuitem">Menu-item 2</li>
      <li class="menuitem">Menu-item 3</li>
      <li class="menuitem">Menu-item 4</li>
      <li class="menuitem">Menu-item 5</li>
    </ul>
  </div>
  <div id="main">

```

```

<h1>Resize the browser window to see the effect!</h1>
<p>This example shows a menu that will float to the left of the page if the viewport is 480 pixels wide or wider. If the viewport is less than 480 pixels, the menu will be on top of the content.</p>
</div>
</div>
</body>
</html>

```

For a full overview of all the media types and features/expressions, please look at the:

https://www.w3schools.com/cssref/css3_pr_mediaquery.php

CSS More Media Query Examples:

https://www.w3schools.com/css/css3_mediaqueries_ex.asp

CSS FlexBox

CSS Flexbox Layout Module

Before the Flexbox Layout module, there were four layout modes:

- Block, for sections in a webpage
- Inline, for text
- Table, for two-dimensional table data
- Positioned, for explicit position of an element

The Flexible Box Layout Module makes it easier to design flexible responsive layout structures without using float or positioning.

- Flexbox Elements:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  background-color: DodgerBlue;
}
.flex-container > div {
  background-color: #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Create a Flex Container</h1>
<div class="flex-container">
  <div>1</div>
  <div>2</div>

```

```
<div>3</div>
</div>
```

<p>A Flexible Layout must have a parent element with the display property set to flex. </p>

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

</body>

</html>

- CSS Flex Container:

~ Parent Element:

The flex container becomes flexible by setting the display property to flex:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  background-color: DodgerBlue;
}
.flex-container > div {
  background-color: #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
```

</style>

</head>

<body>

<h1>Create a Flex Container</h1>

<div class="flex-container">

 <div>1</div>

 <div>2</div>

 <div>3</div>

</div>

<p>A Flexible Layout must have a parent element with the display property set to flex. </p>

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

</body>

</html>

The flex container properties are:

flex-direction

flex-wrap

flex-flow

justify-content

align-items

align-content

~~ The flex-direction Property:

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

eg:

- 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.

eg:

- 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.

eg:

```
.flex-container {
```

```

display: flex;
flex-flow: row wrap; // column wrap.
}

```

~~ The justify-content Property:

The justify-content property is used to align the flex items:

eg:

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

```

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

```

- The flex-start value aligns the flex items at the beginning of the container (this is default):

```

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

```

- 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.

In these examples we use a 200 pixels high container, to better demonstrate the align-items property.

eg:

- 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;
}
```

Note: the example uses different font-size to demonstrate that the items gets aligned by the text baseline:

~~ The align-content Property:

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

In these examples we use a 600 pixels high container, with the flex-wrap property set to wrap, to better demonstrate the align-content property.

eg:

- 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 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 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 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:

To get Perfect Center, Set both the justify-content and align-items properties to center, and the flex item will be perfectly centered:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 300px;
  background-color: DodgerBlue;
}
.flex-container > div {
  background-color: #f1f1f1;
  color: white;
  width: 100px;
  height: 100px;
}
</style>
</head>
<body>
<h1>Perfect Centering</h1>
<p>A flex container with both the justify-content and the align-items properties set to <em>center</em> will align the item(s) in the center (in both axis).</p>
<div class="flex-container">
```

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

Property	Description
<u>align-content</u>	Modifies the behavior of the flex-wrap property. It is similar to align-items, but instead of aligning flex items, it aligns flex lines
<u>align-items</u>	Vertically aligns the flex items when the items do not use all available space on the cross-axis
<u>display</u>	Specifies the type of box used for an HTML element
<u>flex-direction</u>	Specifies the direction of the flexible items inside a flex container
<u>flex-flow</u>	A shorthand property for flex-direction and flex-wrap
<u>flex-wrap</u>	Specifies whether the flex items should wrap or not, if there is not enough room for them on one flex line
<u>justify-content</u>	Horizontally aligns the flex items when the items do not use all available space on the main-axis

- CSS Flex Items:

Child Elements (Items)

The direct child elements of a flex container automatically become flexible (flex) items.

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  background-color: #f1f1f1;
}
.flex-container > div {
  background-color: DodgerBlue;
  color: white;
  width: 100px;
  margin: 10px;
  text-align: center;
  line-height: 75px;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>Flexible Items</h1>
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

<p>All direct children of a flexible container becomes flexible items.</p>

</body>

</html>

The flex item properties are:

order

flex-grow

flex-shrink

flex-basis

flex

align-self

~~ The order Property:

The order property specifies the order of the flex items.

The first flex item in the code does not have to appear as the first item in the layout.

The order value must be a number, default value is 0.

eg:

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.

eg:

Make the third flex item grow eight times faster than the other flex items:

```
<div class="flex-container">
  <div style="flex-grow: 1">1</div>
  <div style="flex-grow: 1">2</div>
  <div style="flex-grow: 8">3</div>
</div>
```

~~ The flex-shrink Property:

The flex-shrink property specifies how much a flex item will shrink relative to the rest of the flex items.

The value must be a number, default value is 1.

eg:

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.

eg:

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.

eg:

Make the third flex item not growable (0), not shrinkable (0), and with an initial length of 200 pixels:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex: 0 0 200px">3</div>
  <div>4</div>
</div>
```

~~ The align-self Property:

The **align-self** property specifies the alignment for the selected item inside the flexible container.

The **align-self** property overrides the default alignment set by the container's **align-items** property.

In these examples we use a 200 pixels high container, to better demonstrate the **align-self** property:

eg:

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>
```

Property	Description
<u>align-self</u>	Specifies the alignment for a flex item (overrides the flex container's align-items property)
<u>flex</u>	A shorthand property for the flex-grow, flex-shrink, and the flex-basis properties
<u>flex-basis</u>	Specifies the initial length of a flex item
<u>flex-grow</u>	Specifies how much a flex item will grow relative to the rest of the flex items inside the same container
<u>flex-shrink</u>	Specifies how much a flex item will shrink relative to the rest of the flex items inside the same container
<u>order</u>	Specifies the order of the flex items inside the same container

- CSS Flex Responsive:

Responsive Flexbox:

We can use media queries to create different layouts for different screen sizes and devices.



- 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):

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
* {
  box-sizing: border-box;
}
.flex-container {
  display: flex;
  flex-direction: row;
  font-size: 30px;
  text-align: center;
}
.flex-item-left {
  background-color: #f1f1f1;
  padding: 10px;
  flex: 50%;
```

```

}

.flex-item-right {
  background-color: dodgerblue;
  padding: 10px;
  flex: 50%;
}

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

</style>
</head>
<body>
<h1>Responsive Flexbox</h1>
<p>The "flex-direction: row;" stacks the flex items horizontally (from left to right).</p>
<p>The "flex-direction: column;" stacks the flex items vertically (from top to bottom).</p>
<p><b>Resize the browser window to see that the direction changes when the screen size is 800px wide or smaller.</b></p>
<div class="flex-container">
  <div class="flex-item-left">1</div>
  <div class="flex-item-right">2</div>
</div>
</body>
</html>

- 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:

eg:
.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%;
  }
}

~~ Responsive Image Gallery Using Flexbox:
<!DOCTYPE html>

```

```
<html>
<style>
* {
  box-sizing: border-box;
}
body {
  margin: 0;
  font-family: Arial;
}
.header {
  text-align: center;
  padding: 32px;
}
.row {
  display: flex;
  flex-wrap: wrap;
  padding: 0 4px;
}
/* Create four equal columns that sits next to each other */
.column {
  flex: 25%;
  max-width: 25%;
  padding: 0 4px;
}
.column img {
  margin-top: 8px;
  vertical-align: middle;
}
/* Responsive layout - makes a two column-layout instead of four columns */
@media (max-width: 800px) {
  .column {
    flex: 50%;
    max-width: 50%;
  }
}
/* Responsive layout - makes the two columns stack on top of each other instead of next to each other */
@media (max-width: 600px) {
  .column {
    flex: 100%;
    max-width: 100%;
  }
}
</style>
<body>
<!-- Header --&gt;
&lt;div class="header"&gt;
  &lt;h1&gt;Responsive Image Gallery&lt;/h1&gt;</pre>
```

```

<p>Resize the browser window to see the responsive effect.</p>
</div>
<!-- Photo Grid -->
<div class="row">
  <div class="column">
    
    
    
    
    
    
    
  </div>
  <div class="column">
    
    
    
    
    
    
  </div>
  <div class="column">
    
    
    
    
    
    
    
  </div>
  <div class="column">
    
    
    
    
    
    
  </div>
</div>
</body>
</html>

```

~~ Responsive Website using Flexbox

Use flexbox to create a responsive website, containing a flexible navigation bar and flexible content:

```

<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>

```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
* {
  box-sizing: border-box;
}
/* Style the body */
body {
  font-family: Arial;
  margin: 0;
}
/* Header/logo Title */
.header {
  padding: 60px;
  text-align: center;
  background: #1abc9c;
  color: white;
}
/* Style the top navigation bar */
.navbar {
  display: flex;
  background-color: #333;
}
/* Style the navigation bar links */
.navbar a {
  color: white;
  padding: 14px 20px;
  text-decoration: none;
  text-align: center;
}
/* Change color on hover */
.navbar a:hover {
  background-color: #ddd;
  color: black;
}
/* Column container */
.row {
  display: flex;
  flex-wrap: wrap;
}
/* Create two unequal columns that sits next to each other */
/* Sidebar/left column */
.side {
  flex: 30%;
  background-color: #f1f1f1;
  padding: 20px;
}
/* Main column */
```

```

.main {
  flex: 70%;
  background-color: white;
  padding: 20px;
}
/* Fake image, just for this example */
.fakeimg {
  background-color: #aaa;
  width: 100%;
  padding: 20px;
}
/* Footer */
.footer {
  padding: 20px;
  text-align: center;
  background: #ddd;
}
/* Responsive layout - when the screen is less than 700px wide, make the two columns
stack on top of each other instead of next to each other */
@media screen and (max-width: 700px) {
  .row, .navbar {
    flex-direction: column;
  }
}
</style>
</head>
<body>
<!-- Note --&gt;
&lt;div style="background:yellow;padding:5px"&gt;
  &lt;h4 style="text-align:center"&gt;Resize the browser window to see the responsive
  effect.&lt;/h4&gt;
&lt;/div&gt;
<!-- Header --&gt;
&lt;div class="header"&gt;
  &lt;h1&gt;My Website&lt;/h1&gt;
  &lt;p&gt;With a &lt;b&gt;flexible&lt;/b&gt; layout.&lt;/p&gt;
&lt;/div&gt;
<!-- Navigation Bar --&gt;
&lt;div class="navbar"&gt;
  &lt;a href="#"&gt;Link&lt;/a&gt;
  &lt;a href="#"&gt;Link&lt;/a&gt;
  &lt;a href="#"&gt;Link&lt;/a&gt;
  &lt;a href="#"&gt;Link&lt;/a&gt;
&lt;/div&gt;
<!-- The flexible grid (content) --&gt;
&lt;div class="row"&gt;
  &lt;div class="side"&gt;
    &lt;h2&gt;About Me&lt;/h2&gt;
</pre>

```

```
<h5>Photo of me:</h5>
<div class="fakeimg" style="height:200px;">Image</div>
<p>Some text about me in culpa qui officia deserunt mollit anim..</p>
<h3>More Text</h3>
<p>Lorem ipsum dolor sit ame.</p>
<div class="fakeimg" style="height:60px;">Image</div><br>
<div class="fakeimg" style="height:60px;">Image</div><br>
<div class="fakeimg" style="height:60px;">Image</div>
</div>
<div class="main">
    <h2>TITLE HEADING</h2>
    <h5>Title description, Dec 7, 2017</h5>
    <div class="fakeimg" style="height:200px;">Image</div>
    <p>Some text..</p>
    <p>Sunt in culpa qui officia deserunt mollit anim id est laborum consectetur adipisciing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco.</p>
    <br>
    <h2>TITLE HEADING</h2>
    <h5>Title description, Sep 2, 2017</h5>
    <div class="fakeimg" style="height:200px;">Image</div>
    <p>Some text..</p>
    <p>Sunt in culpa qui officia deserunt mollit anim id est laborum consectetur adipisciing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco.</p>
    </div>
</div>
<!-- Footer -->
<div class="footer">
    <h2>Footer</h2>
</div>
</body>
</html>
```

* * * * *

CSS RESPONSIVE WEB DESIGN (RWD)

- **CSS Responsive Web Design:**

Responsive Web Design - Introduction

What is Responsive Web Design?

Responsive web design makes your web page look good on all devices.

Responsive web design uses only HTML and CSS.

Responsive web design is not a program or a JavaScript.

~~ Designing For The Best Experience For All Users:

Web pages can be viewed using many different devices: desktops, tablets, and phones.

Your web page should look good, and be easy to use, regardless of the device.

Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:



It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

eg:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}
.row::after {
  content: "";
  clear: both;
  display: table;
}
[class*="col-"] {
  float: left;
  padding: 15px;
}
html {
```

```

font-family: "Lucida Sans", sans-serif;
}
.header {
background-color: #9933cc;
color: #ffffff;
padding: 15px;
}
.menu ul {
list-style-type: none;
margin: 0;
padding: 0;
}
.menu li {
padding: 8px;
margin-bottom: 7px;
background-color: #33b5e5;
color: #ffffff;
box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
background-color: #0099cc;
}
.aside {
background-color: #33b5e5;
padding: 15px;
color: #ffffff;
text-align: center;
font-size: 14px;
box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.footer {
background-color: #0099cc;
color: #ffffff;
text-align: center;
font-size: 12px;
padding: 15px;
}
/* For mobile phones: */
[class*="col-"] {
width: 100%;
}
@media only screen and (min-width: 600px) {
/* For tablets: */
.col-s-1 {width: 8.33%;}
.col-s-2 {width: 16.66%;}
.col-s-3 {width: 25%;}
.col-s-4 {width: 33.33%;}
.col-s-5 {width: 41.66%;}
}

```

```

.col-s-6 {width: 50%;}
.col-s-7 {width: 58.33%;}
.col-s-8 {width: 66.66%;}
.col-s-9 {width: 75%;}
.col-s-10 {width: 83.33%;}
.col-s-11 {width: 91.66%;}
.col-s-12 {width: 100%;}
}
@media only screen and (min-width: 768px) {
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
}

```

</style>

</head>

<body>

<div class="header">

<h1>Chania</h1>

</div>

<div class="row">

<div class="col-3 col-s-3 menu">

- The Flight
- The City
- The Island
- The Food

</div>

<div class="col-6 col-s-9">

The City

<p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>

</div>

<div class="col-3 col-s-12">

What?

<p>Chania is a city on the island of Crete.</p>

Where?

```

<p>Crete is a Greek island in the Mediterranean Sea.</p>
<h2>How?</h2>
<p>You can reach Chania airport from all over Europe.</p>
</div>
</div>
</div>
<div class="footer">
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>
</body>
</html>

```

Responsive Web Design - The Viewport

What is The Viewport?

The viewport is the user's visible area of a web page.

The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.

Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size.

Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen.

This was not perfect!! But a quick fix.

- Setting the Viewport:

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

You should include the following <meta> viewport element in all your web pages:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

This gives the browser instructions on how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

- Size Content to The Viewport:

Users are used to scroll websites vertically on both desktop and mobile devices - but not horizontally!

So, if the user is forced to scroll horizontally, or zoom out, to see the whole web page it results in a poor user experience.

Some additional rules to follow:

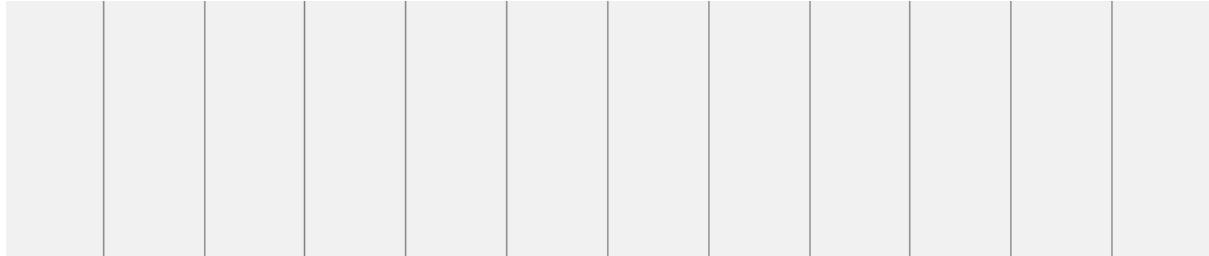
1. Do NOT use large fixed width elements - For example, if an image is displayed at a width wider than the viewport it can cause the viewport to scroll horizontally. Remember to adjust this content to fit within the width of the viewport.
2. Do NOT let the content rely on a particular viewport width to render well - Since screen dimensions and width in CSS pixels vary widely between devices, content should not rely on a particular viewport width to render well.
3. Use CSS media queries to apply different styling for small and large screens - Setting large absolute CSS widths for page elements will cause the element to be too wide for the

viewport on a smaller device. Instead, consider using relative width values, such as width: 100%. Also, be careful of using large absolute positioning values. It may cause the element to fall outside the viewport on small devices.

Responsive Web Design - Grid-View

What is a Grid-View?

Many web pages are based on a grid-view, which means that the page is divided into columns:



Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.



A responsive grid-view often has 12 columns, and has a total width of 100%, and will shrink and expand as you resize the browser window.

- Building a Responsive Grid-View:

Let's start building a responsive grid-view.

First ensure that all HTML elements have the box-sizing property set to border-box. This makes sure that the padding and border are included in the total width and height of the elements.

Add the following code in your CSS:

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

eg:
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
```

```

}
.header {
  border: 1px solid red;
  padding: 15px;
}
.menu {
  width: 25%;
  float: left;
  padding: 15px;
  border: 1px solid red;
}
.main {
  width: 75%;
  float: left;
  padding: 15px;
  border: 1px solid red;
}

```

</style>

</head>

<body>

<div class="header">

<h1>Chania</h1>

</div>

<div class="menu">

The Flight

The City

The Island

The Food

</div>

<div class="main">

<h1>The City</h1>

<p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>

<p>Resize the browser window to see how the content respond to the resizing.</p>

</div>

</body>

</html>

The example above is fine if the web page only contains two columns. However, we want to use a responsive grid-view with 12 columns, to have more control over the web page.

First we must calculate the percentage for one column: $100\% / 12 \text{ columns} = 8.33\%$. Then we make one class for each of the 12 columns, class="col-" and a number defining how many columns the section should span:

CSS:

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
```

```
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

All these columns should be floating to the left, and have a padding of 15px:

CSS:

```
[class*="col-"] {
  float: left;
  padding: 15px;
  border: 1px solid red;
}
```

Each row should be wrapped in a <div>. The number of columns inside a row should always add up to 12:

HTML:

```
<div class="row">
  <div class="col-3">...</div> <!-- 25% -->
  <div class="col-9">...</div> <!-- 75% -->
</div>
```

The columns inside a row are all floating to the left, and are therefore taken out of the flow of the page, and other elements will be placed as if the columns do not exist. To prevent this, we will add a style that clears the flow:

CSS:

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

We also want to add some styles and colors to make it look better:

```
html {
  font-family: "Lucida Sans", sans-serif;
}

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}

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

.menu li {
```

```

padding: 8px;
margin-bottom: 7px;
background-color :#33b5e5;
color: #ffffff;
box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
background-color: #0099cc;
}
# Notice that the webpage in the example does not look good when you resize the browser
window to a very small width. In the next chapter you will learn how to fix that.

```

Responsive Web Design - Media Queries

What is a Media Query?

Media query is a CSS technique introduced in CSS3.

It uses the @media rule to include a block of CSS properties only if a certain condition is true.

eg:

If the browser window is 600px or smaller, the background color will be lightblue:

```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
background-color: lightgreen;
}
@media only screen and (max-width: 600px) {
body {
background-color: lightblue;
}
}
</style>
</head>
<body>
<p>Resize the browser window. When the width of this document is 600 pixels or less, the
background-color is "lightblue", otherwise it is "lightgreen".</p>
</body>
</html>

```

- Add a BreakPoint:

Earlier in this tutorial we made a web page with rows and columns, and it was responsive, but it did not look good on a small screen.

Media queries can help with that. We can add a breakpoint where certain parts of the design will behave differently on each side of the breakpoint.

eg:

When the screen (browser window) gets smaller than 768px, each column should have a width of 100%:

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
    /* For mobile phones: */
    [class*="col-"] {
        width: 100%;
    }
}
```

- Always Design for Mobile First:

Mobile First means designing for mobile before designing for desktop or any other device (This will make the page display faster on smaller devices).

This means that we must make some changes in our CSS.

Instead of changing styles when the width gets smaller than 768px, we should change the design when the width gets larger than 768px. This will make our design Mobile First:

eg:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
    box-sizing: border-box;
}
.row::after {
    content: "";
    clear: both;
    display: table;
}
[class*="col-"] {
    float: left;
    padding: 15px;
}
html {
    font-family: "Lucida Sans", sans-serif;
}
```

```

.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}
.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
  background-color: #0099cc;
}
.aside {
  background-color: #33b5e5;
  padding: 15px;
  color: #ffffff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.footer {
  background-color: #0099cc;
  color: #ffffff;
  text-align: center;
  font-size: 12px;
  padding: 15px;
}
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
}

```

```

.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
}
</style>
</head>
<body>
<div class="header">
  <h1>Chania</h1>
</div>
<div class="row">
  <div class="col-3 menu">
    <ul>
      <li>The Flight</li>
      <li>The City</li>
      <li>The Island</li>
      <li>The Food</li>
    </ul>
  </div>
  <div class="col-6">
    <h1>The City</h1>
    <p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>
  </div>
  <div class="col-3 right">
    <div class="aside">
      <h2>What?</h2>
      <p>Chania is a city on the island of Crete.</p>
      <h2>Where?</h2>
      <p>Crete is a Greek island in the Mediterranean Sea.</p>
      <h2>How?</h2>
      <p>You can reach Chania airport from all over Europe.</p>
    </div>
  </div>
</div>
<div class="footer">
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>
</body>
</html>

```

• Another Breakpoint:

You can add as many breakpoints as you like.

We will also insert a breakpoint between tablets and mobile phones

We do this by adding one more media query (at 600px), and a set of new classes for devices larger than 600px (but smaller than 768px):

```
# Note that the two sets of classes are almost identical, the only difference is the name (col- and col-s-):
eg:
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}
@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
```

It might seem odd that we have two sets of identical classes, but it gives us the opportunity in HTML, to decide what will happen with the columns at each breakpoint:

- [HTML Example](#):

For desktop:

The first and the third section will both span 3 columns each. The middle section will span 6 columns.

For tablets:

The first section will span 3 columns, the second will span 9, and the third section will be displayed below the first two sections, and it will span 12 columns:

eg:

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}
.row::after {
  content: "";
  clear: both;
  display: table;
}
[class*="col-"] {
  float: left;
  padding: 15px;
}
html {
  font-family: "Lucida Sans", sans-serif;
}
.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}
.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #33b5e5;
  color: #ffffff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
  background-color: #0099cc;
}
.aside {
  background-color: #33b5e5;
  padding: 15px;
  color: #ffffff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.footer {
```

```

background-color: #0099cc;
color: #ffffff;
text-align: center;
font-size: 12px;
padding: 15px;
}
/* For mobile phones: */
[class*="col-"] {
  width: 100%;
}
@media only screen and (min-width: 600px) {
  /* For tablets: */
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}
@media only screen and (min-width: 768px) {
  /* For desktop: */
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
</style>
</head>
<body>
<div class="header">
  <h1>Chania</h1>
</div>
<div class="row">
  <div class="col-3 col-s-3 menu">

```

```

<ul>
  <li>The Flight</li>
  <li>The City</li>
  <li>The Island</li>
  <li>The Food</li>
</ul>
</div>
<div class="col-6 col-s-9">
  <h1>The City</h1>
  <p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>
</div>
<div class="col-3 col-s-12">
  <div class="aside">
    <h2>What?</h2>
    <p>Chania is a city on the island of Crete.</p>
    <h2>Where?</h2>
    <p>Crete is a Greek island in the Mediterranean Sea.</p>
    <h2>How?</h2>
    <p>You can reach Chania airport from all over Europe.</p>
  </div>
</div>
</div>
<div class="footer">
  <p>Resize the browser window to see how the content respond to the resizing.</p>
</div>
</body>
</html>

```

• Typical Device Breakpoints:

There are tons of screens and devices with different heights and widths, so it is hard to create an exact breakpoint for each device. To keep things simple you could target five groups:

eg:

```

/* Extra small devices (phones, 600px and down) */
@media only screen and (max-width: 600px) {...}
/* Small devices (portrait tablets and large phones, 600px and up) */
@media only screen and (min-width: 600px) {...}
/* Medium devices (landscape tablets, 768px and up) */
@media only screen and (min-width: 768px) {...}
/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {...}
/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {...}

```

• Orientation: Portrait / Landscape:

Media queries can also be used to change the 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:

eg:

The web page will have a lightblue background if the orientation is in landscape mode:

```
@media only screen and (orientation: landscape) {
  body {
    background-color: lightblue;
  }
}
```

- Hide Elements With Media Queries:

Another common use of media queries is to hide elements on different screen sizes.

eg:

```
/* If the screen size is 600px wide or less, hide the element */
@media only screen and (max-width: 600px) {
  div.example {
    display: none;
  }
}
```

- Change Font Size With Media Queries:

You can also use media queries to change the font size of an element on different screen sizes.

eg:

* If the screen size is 601px or more, set the font-size of <div> to 80px */

```
@media only screen and (min-width: 601px) {
  div.example {
    font-size: 80px;
  }
}
```

* If the screen size is 600px or less, set the font-size of <div> to 30px */

```
@media only screen and (max-width: 600px) {
  div.example {
    font-size: 30px;
  }
}
```

Responsive Web Design - Images

- Using The width Property:

If the width property is set to a percentage and the height property is set to "auto", the image will be responsive and scale up and down:

eg:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
img {
  width: 100%;
  height: auto;
}
```

```

</style>
</head>
<body>


Resize the browser window to see how the image will scale.


</body>
</html>
# Notice: that in the example above, the image can be scaled up to be larger than its
original size. A better solution, in many cases, will be to use the max-width property instead.
• Using The max-width Property:

```

If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

eg:

```

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

```

~ Add an Image to The Example Web Page:

eg:

```

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

```

• Background Images:

Background images can also respond to resizing and scaling.

Here we will show three different methods:

1. If the background-size property is set to "contain", the background image will scale, and try to fit the content area. However, the image will keep its aspect ratio (the proportional relationship between the image's width and height):

eg:

```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
div {
  width: 100%;
  height: 400px;
  background-image: url('img_flowers.jpg');
  background-repeat: no-repeat;
  background-size: contain;
  border: 1px solid red;
}
</style>
</head>
<body>


Resize the browser window to see the effect.


<div></div>

```

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

2. If the background-size property is set to "100% 100%", the background image will stretch to cover the entire content area:

eg:

```
div {
    width: 100%;
    height: 400px;
    background-image: url('img_flowers.jpg');
    background-size: 100% 100%;
    border: 1px solid red;
}
```

3. If the background-size property is set to "cover", the background image will scale to cover the entire content area. Notice that the "cover" value keeps the aspect ratio, and some part of the background image may be clipped:

eg:

```
div {
    width: 100%;
    height: 400px;
    background-image: url('img_flowers.jpg');
    background-size: cover;
    border: 1px solid red;
}
```

- Different Images for Different Devices:

A large image can be perfect on a big computer screen, but useless on a small device. Why load a large image when you have to scale it down anyway? To reduce the load, or for any other reasons, you can use media queries to display different images on different devices.

eg:

```
/* For width smaller than 400px: */
body {
    background-image: url('img_smallflower.jpg');
}

/* For width 400px and larger: */
@media only screen and (min-width: 400px) {
    body {
        background-image: url('img_flowers.jpg');
    }
}
```

~ You can use the media query min-device-width, instead of min-width, which checks the device width, instead of the browser width. Then the image will not change when you resize the browser window:

```
/* For devices smaller than 400px: */
body {
    background-image: url('img_smallflower.jpg');
}

/* For devices 400px and larger: */
@media only screen and (min-device-width: 400px) {
    body {
```

```

background-image: url('img_flowers.jpg');
}
}

```

- The HTML <picture> Element:

The HTML <picture> element gives web developers more flexibility in specifying image resources.

The most common use of the <picture> element will be for images used in responsive designs. Instead of having one image that is scaled up or down based on the viewport width, multiple images can be designed to more nicely fill the browser viewport.

The <picture> element works similar to the <video> and <audio> elements. You set up different sources, and the first source that fits the preferences is the one being used:

eg:

```

<picture>
  <source srcset="img_smallflower.jpg" media="(max-width: 400px)">
  <source srcset="img_flowers.jpg">
    
</picture>

```

The srcset attribute is required, and defines the source of the image.

The media attribute is optional, and accepts the media queries.

You should also define an element for browsers that do not support the <picture> element.

Responsive Web Design - Videos

- Using The width Property:

If the width property is set to 100%, the video player will be responsive and scale up and down:

eg:

```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
video {
  width: 100%;
  height: auto;
}
</style>
</head>
<body>
<video width="400" controls>
  <source src="mov_bbb.mp4" type="video/mp4">
  <source src="mov_bbb.ogg" type="video/ogg">
  Your browser does not support HTML5 video.
</video>
<p>Resize the browser window to see how the size of the video player will scale.</p>

```

```
</body>
</html>
# Notice: that in the example above, the video player can be scaled up to be larger than its
original size. A better solution, in many cases, will be to use the max-width property instead.
```

- Using The max-width Property:

If the max-width property is set to 100%, the video player will scale down if it has to, but never scale up to be larger than its original size:

eg:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
video {
  max-width: 100%;
  height: auto;
}
</style>
</head>
<body>
<video width="400" controls>
  <source src="mov_bbb.mp4" type="video/mp4">
  <source src="mov_bbb.ogv" type="video/ogg">
  Your browser does not support HTML5 video.
</video>
<p>Resize the browser window to see how the size of the video player will scale when the
width is less than 400px.</p>
</body>
</html>
```

- Add a Video to the Example Web Page:

We want to add a video in our example web page. The video will be resized to always take up all the available space:

eg:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}
video {
  width: 100%;
  height: auto;
}
.row:after {
  content: "";
  clear: both;
```

```
display: table;
}
[class*="col-"] {
  float: left;
  padding: 15px;
  width: 100%;
}
@media only screen and (min-width: 600px) {
  .col-s-1 {width: 8.33%;}
  .col-s-2 {width: 16.66%;}
  .col-s-3 {width: 25%;}
  .col-s-4 {width: 33.33%;}
  .col-s-5 {width: 41.66%;}
  .col-s-6 {width: 50%;}
  .col-s-7 {width: 58.33%;}
  .col-s-8 {width: 66.66%;}
  .col-s-9 {width: 75%;}
  .col-s-10 {width: 83.33%;}
  .col-s-11 {width: 91.66%;}
  .col-s-12 {width: 100%;}
}
@media only screen and (min-width: 768px) {
  .col-1 {width: 8.33%;}
  .col-2 {width: 16.66%;}
  .col-3 {width: 25%;}
  .col-4 {width: 33.33%;}
  .col-5 {width: 41.66%;}
  .col-6 {width: 50%;}
  .col-7 {width: 58.33%;}
  .col-8 {width: 66.66%;}
  .col-9 {width: 75%;}
  .col-10 {width: 83.33%;}
  .col-11 {width: 91.66%;}
  .col-12 {width: 100%;}
}
html {
  font-family: "Lucida Sans", sans-serif;
}
.header {
  background-color: #9933cc;
  color: #ffffff;
  padding: 15px;
}
.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

```

.menu li {
    padding: 8px;
    margin-bottom: 7px;
    background-color: #33b5e5;
    color: #ffffff;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.menu li:hover {
    background-color: #0099cc;
}
.aside {
    background-color: #33b5e5;
    padding: 15px;
    color: #ffffff;
    text-align: center;
    font-size: 14px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}
.footer {
    background-color: #0099cc;
    color: #ffffff;
    text-align: center;
    font-size: 12px;
    padding: 15px;
}

```

</style>

</head>

<body>

<div class="header">

<h1>Chania</h1>

</div>

<div class="row">

<div class="col-3 col-s-3 menu">

The Flight

The City

The Island

The Food

</div>

<div class="col-6 col-s-9">

<h1>The City</h1>

<p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>

<video width="400" controls>

<source src="mov_bbb.mp4" type="video/mp4">

<source src="mov_bbb.ogg" type="video/ogg">

Your browser does not support HTML5 video.

```

</video>
</div>
<div class="col-3 col-s-12">
  <div class="aside">
    <h2>What?</h2>
    <p>Chania is a city on the island of Crete.</p>
    <h2>Where?</h2>
    <p>Crete is a Greek island in the Mediterranean Sea.</p>
    <h2>How?</h2>
    <p>You can reach Chania airport from all over Europe.</p>
  </div>
</div>
</div>
<div class="footer">
  <p>Resize the browser window to see how the content responds to the resizing.</p>
</div>
</body>
</html>

```

CSS Templates:

For CSS Templates visit this link:

https://www.w3schools.com/css/css_rwd_templates.asp

CSS Grid Layout Module

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.

eg:

```

<!DOCTYPE html>
<html>
<head>
<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>
</body>
</html>

```

• Display Property:

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

eg:

```

<!DOCTYPE html>
<html>
<head>
<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>display: grid</h1>
<p>Use display: grid; to make a block-level grid container:</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>
</body>
</html>
```

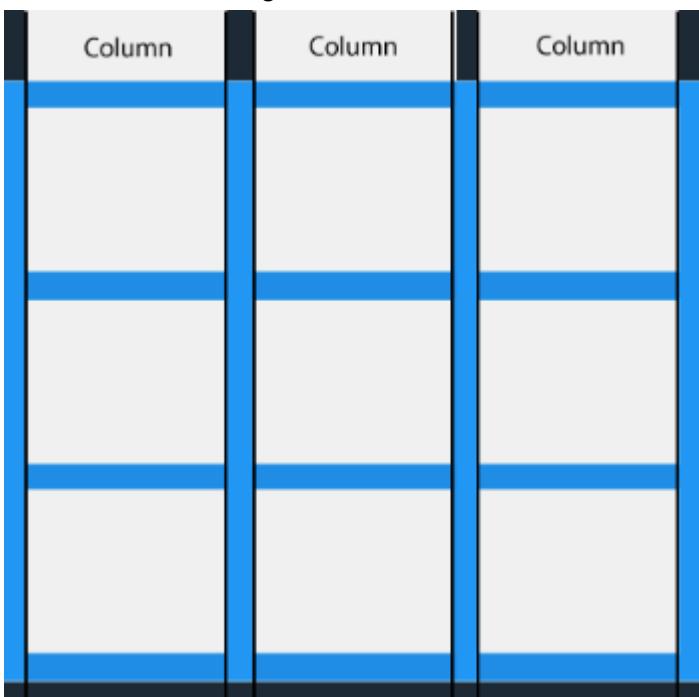
eg 2:

```
.grid-container {
  display: inline-grid;
}
```

All direct children of the grid container automatically become grid items.

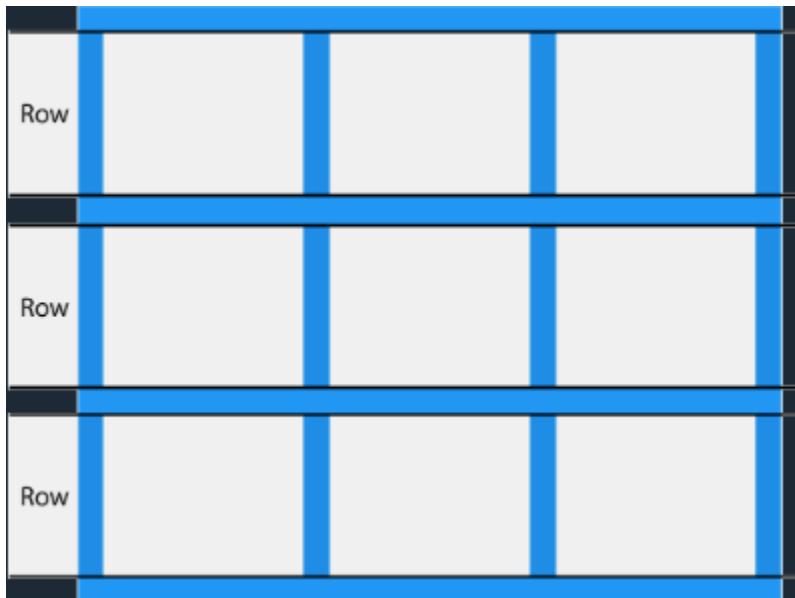
- Grid Columns:

The vertical lines of grid items are called columns.



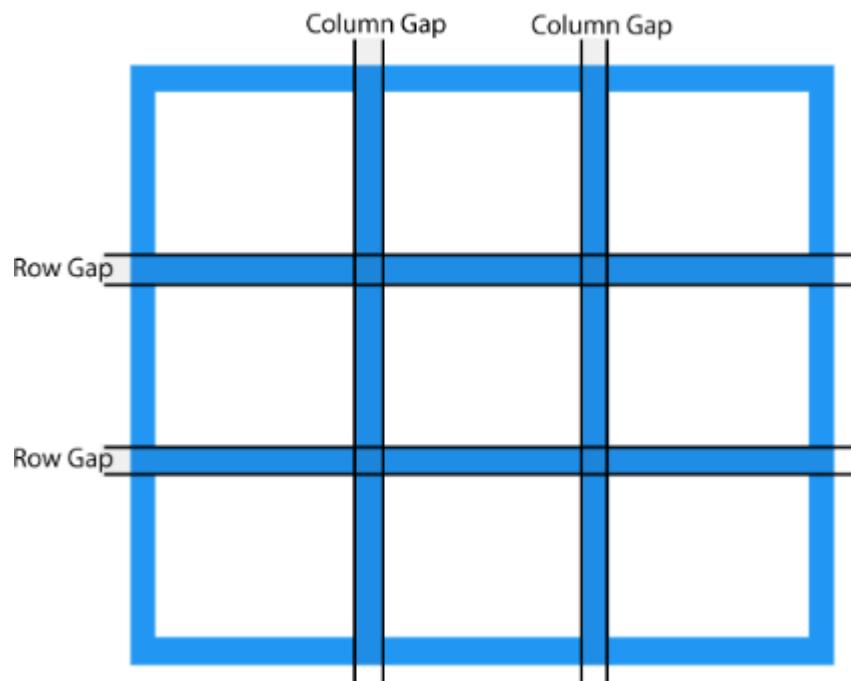
- Grid Rows:

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

eg:

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

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```

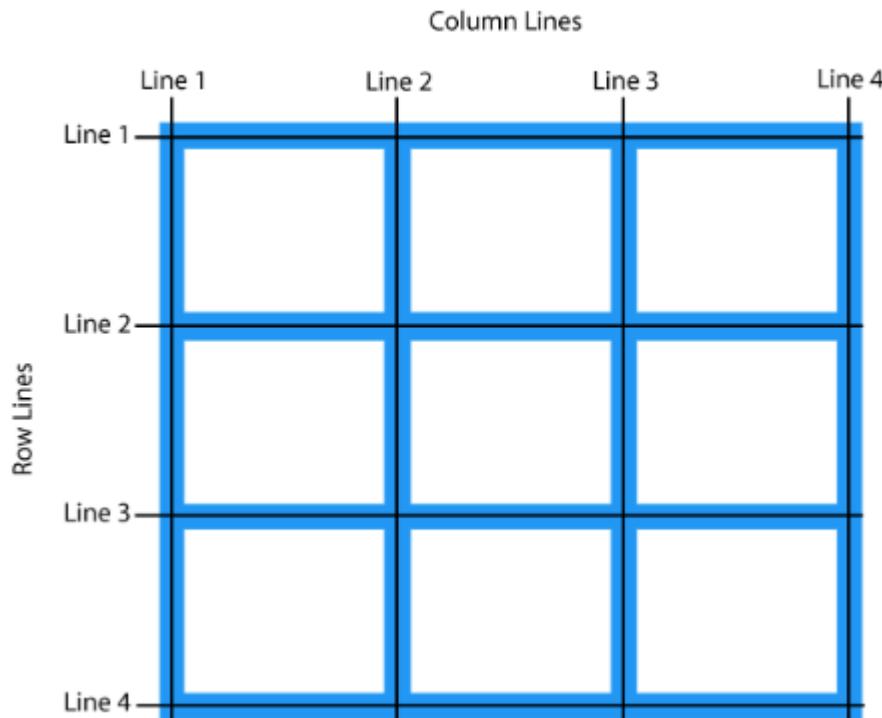
.grid-container {
  display: grid;
  column-gap: 50px;
  grid-template-columns: auto auto auto;
  background-color: #2196F3;
  padding: 10px;
}
.grid-item {
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>
<h1>The column-gap Property</h1>
<p>Use the <em>column-gap</em> property to adjust the space between the columns:</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>
</body>
</html>
- 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:

eg:

Place a grid item at column line 1, and let it end on column line 3:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
</style>
```

```

</head>
<body>
<h1>Grid Lines</h1>
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
</div>
<p>You can refer to line numbers when placing grid items.</p>
</body>
</html>

```

- 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;
}

```

- All CSS Grid Properties:

Property	Description
<u>column-gap</u>	Specifies the gap between the columns
<u>gap</u>	A shorthand property for the <i>row-gap</i> and the <i>column-gap</i> properties
<u>grid</u>	A shorthand property for the <i>grid-template-rows</i> , <i>grid-template-columns</i> , <i>grid-template-areas</i> , <i>grid-auto-rows</i> , <i>grid-auto-columns</i> , and the <i>grid-auto-flow</i> properties
<u>grid-area</u>	Either specifies a name for the grid item, or this property is a shorthand property for the <i>grid-row-start</i> , <i>grid-column-start</i> , <i>grid-row-end</i> , and <i>grid-column-end</i> properties
<u>grid-auto-columns</u>	Specifies a default column size
<u>grid-auto-flow</u>	Specifies how auto-placed items are inserted in the grid
<u>grid-auto-rows</u>	Specifies a default row size
<u>grid-column</u>	A shorthand property for the <i>grid-column-start</i> and the <i>grid-column-end</i> properties
<u>grid-column-end</u>	Specifies where to end the grid item
<u>grid-column-gap</u>	Specifies the size of the gap between columns
<u>grid-column-start</u>	Specifies where to start the grid item

<u>grid-gap</u>	A shorthand property for the <i>grid-row-gap</i> and <i>grid-column-gap</i> properties
<u>grid-row</u>	A shorthand property for the <i>grid-row-start</i> and the <i>grid-row-end</i> properties
<u>grid-row-end</u>	Specifies where to end the grid item
<u>grid-row-gap</u>	Specifies the size of the gap between rows
<u>grid-row-start</u>	Specifies where to start the grid item
<u>grid-template</u>	A shorthand property for the <i>grid-template-rows</i> , <i>grid-template-columns</i> and <i>grid-areas</i> properties
<u>grid-template-areas</u>	Specifies how to display columns and rows, using named grid items
<u>grid-template-columns</u>	Specifies the size of the columns, and how many columns in a grid layout
<u>grid-template-rows</u>	Specifies the size of the rows in a grid layout
<u>row-gap</u>	Specifies the gap between the grid rows

CSS 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.

eg:

Make a grid with 4 columns:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
```

```

padding: 20px 0;
font-size: 30px;
}
</style>
</head>
<body>
<h1>The grid-template-columns Property</h1>
<p>You can use the <em>grid-template-columns</em> property to specify the number of columns in your grid layout.</p>
<div class="grid-container">
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
<div>5</div>
<div>6</div>
<div>7</div>
<div>8</div>
</div>
</body>
</html>

```

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.

```
.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.

1	2	3	4
5	6	7	8

eg:

The value is a space-separated-list, where each value defines the height of the respective row:

```
<!DOCTYPE html>
```

```

<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  grid-template-rows: 80px 200px;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>The grid-template-rows Property</h1>
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
<p>Use the <em>grid-template-rows</em> property to specify the size (height) of each row.</p>
</body>
</html>
# Note: The grid's total width has to be less than the container's width for the justify-content property to have any effect.
eg:
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  justify-content: space-evenly;
  grid-template-columns: 50px 50px 50px; /*Make the grid smaller than the container*/
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

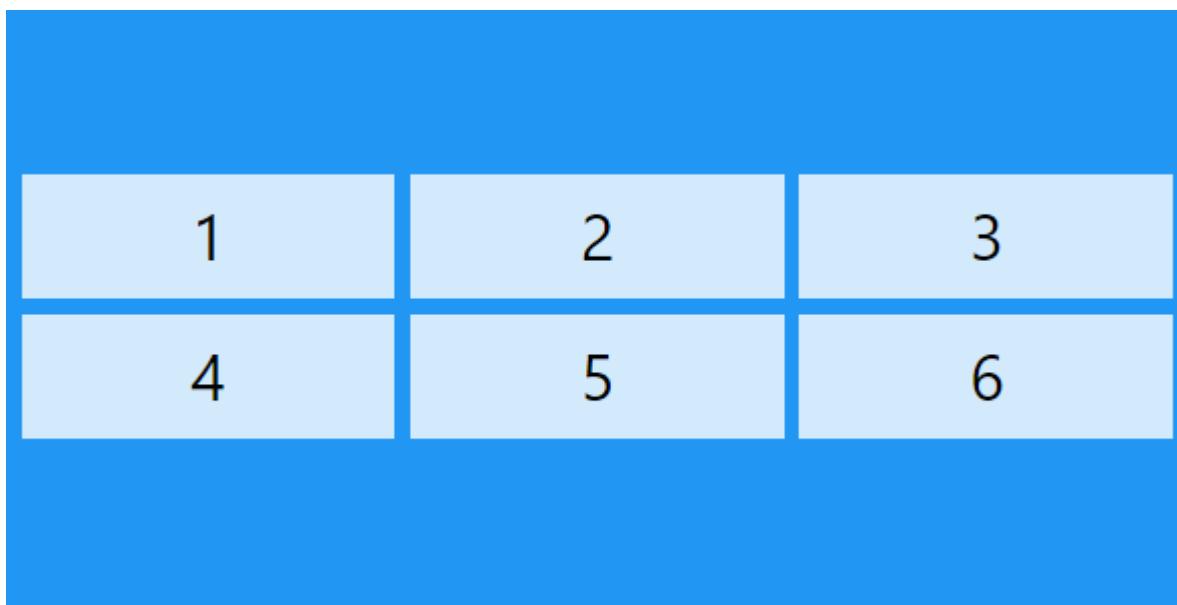
```

```

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>The justify-content Property</h1>
<p>Use the <em>justify-content</em> property to align the grid inside the container.</p>
<p>The value "space-evenly" will give the columns equal amount of space between, and
around them:</p>
<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
</body>
</html>
~ .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.

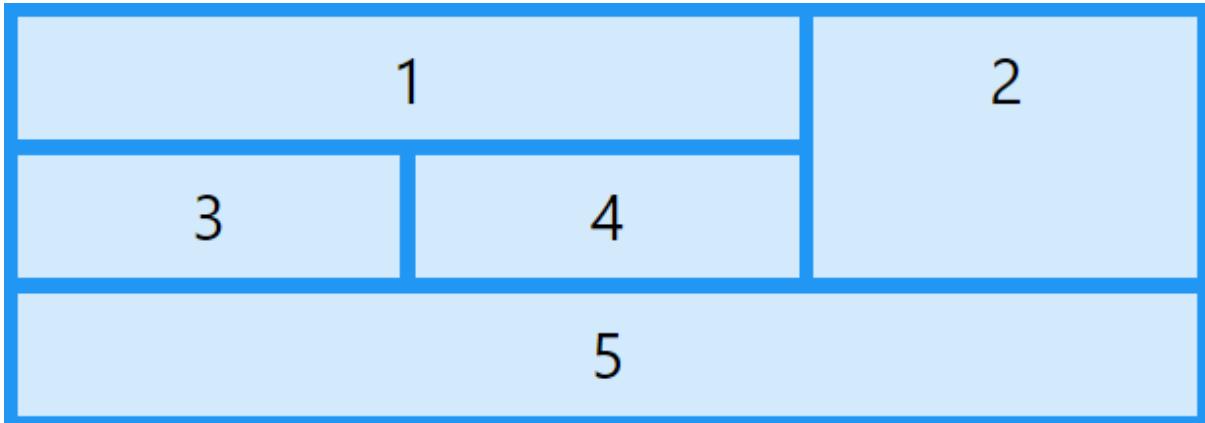


Note: The grid's total height has to be less than the container's height for the align-content property to have any effect.

eg:

```
~ .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;}
```

CSS Grid Item



- Child Elements (Items)

A grid container contains grid items.

By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

- The grid-column Property:

The grid-column property defines which column(s) to place an item.

You define where the item will start, and where the item will end.



Note: The grid-column property is a shorthand property for the grid-column-start and the grid-column-end properties.

~~ To place an item, you can refer to line numbers, or use the keyword "span" to define how many columns the item will span.

eg:

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

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
```

```

.grid-container > div {
background-color: rgba(255, 255, 255, 0.8);
text-align: center;
padding: 20px 0;
font-size: 30px;
}
.item1 {
grid-column: 1 / 5;
}
</style>
</head>
<body>
<h1>The grid-column Property</h1>
<p>Use the <em> grid-column</em> property to specify where to place an item.</p>
<p>Item1 will start on column 1 and end before column 5:</p>
<div class="grid-container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
<div class="item6">6</div>
<div class="item7">7</div>
<div class="item8">8</div>
<div class="item9">9</div>
<div class="item10">10</div>
<div class="item11">11</div>
<div class="item12">12</div>
<div class="item13">13</div>
<div class="item14">14</div>
<div class="item15">15</div>
</div>
</body>
</html>

```

~ 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.

1	2	3	4	5	6
7	8	9	10	11	
12	13	14	15	16	

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:

eg:

~ 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.

1		8		2	
3				4	
5				6	
7				9	
10	11	12	13	14	15

eg:

~ Make "item8" start on row-line 1 and column-line 2, and end on row-line 5 and column line 6:

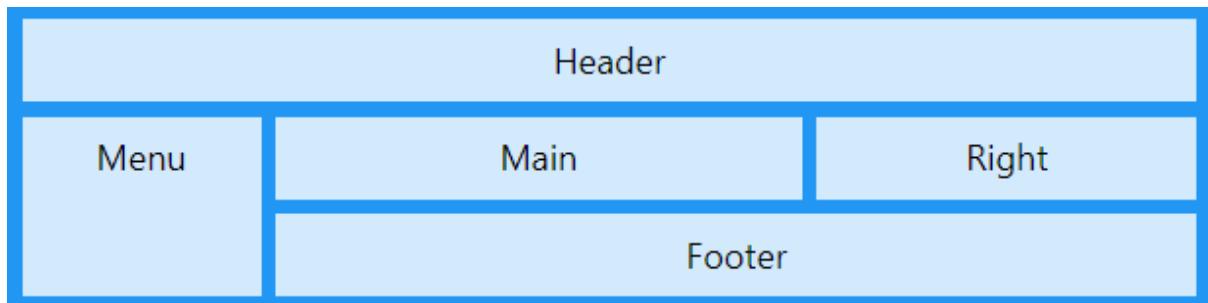
```
.item8 {
  grid-area: 1 / 2 / 5 / 6;
}
```

~ 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.

eg:

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 are defined inside the apostrophes, separated by a space.

Note: A period sign represents a grid item with no name.

eg:

~ 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 . . .';
}
```

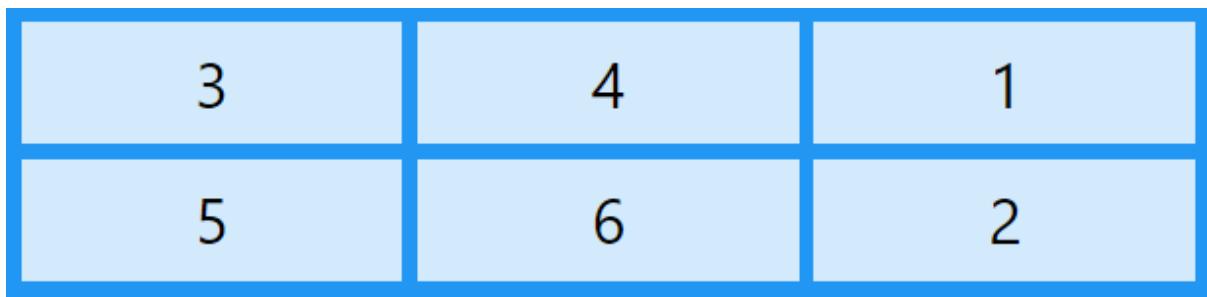
~ Name all items, and make a ready-to-use webpage template:

```
<!DOCTYPE html>
<html>
<head>
<style>
.item1 { grid-area: header; }
.item2 { grid-area: menu; }
.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }
.grid-container {
  display: grid;
  grid-template-areas:
    'header header header header header header'
    'menu main main main right right'
    'menu footer footer footer footer footer';
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>
<h1>The grid-area Property</h1>
<p>You can use the <em> grid-area</em> property to name grid items.</p>
<p>You can refer to the name when you set up the grid layout, by using the <em> grid-template-areas</em> property on the grid container.</p>
<p>This grid layout contains six columns and three rows:</p>
<div class="grid-container">
  <div class="item1">Header</div>
  <div class="item2">Menu</div>
  <div class="item3">Main</div>
  <div class="item4">Right</div>
  <div class="item5">Footer</div>
</div>
</body>
</html>
```

- 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.



eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
.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; }
</style>
</head>
<body>
<h1>Sort the Items</h1>
<p>The grid items do not have to be displayed in the same order as they are written in the HTML code.</p>
<div class="grid-container">
<div class="item1">1</div>
<div class="item2">2</div>
<div class="item3">3</div>
<div class="item4">4</div>
<div class="item5">5</div>
<div class="item6">6</div>
</div>
</body>
</html>
```

You can rearrange the order for certain screen sizes, by using media queries:

eg:

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
@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; }
}
</style>
</head>
<body>
<h1>Rearrange the Order on Small Devices</h1>
<p>Resize the window to 500 pixels to see the effect.</p>
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
</div>
</body>
</html>
```

❤️ ALL IZZ WELL ❤️

By:
– N.JASHWANTH.