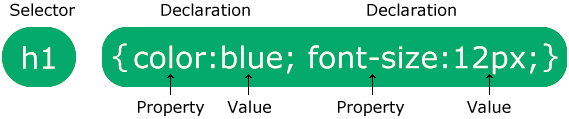
* CSS stands for Cascading Style Sheets
* CSS describes how HTML elements are to be displayed on screen, paper, or in other media
* CSS saves a lot of work. It can control the layout of multiple web pages all at once
* External stylesheets are stored in CSS files

Example:

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

syntax:



CSS Selectors

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

We can divide CSS selectors into five categories:

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

## **The CSS element Selector**

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

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

## **The CSS id Selector**

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

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

## **The CSS class Selector**

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

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

## **The CSS Universal Selector**

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

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

## **The CSS Grouping Selector**

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

# 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;  
}

/\* 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;  
}

CSS 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

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

in the paragraph the first line will be highlight in red with small caps

example : ::first-letter, ::before, ::after, ::selection

|  |  |  |
| --- | --- | --- |
| **Selector** | **Example** | **Example description** |
| [::after](https://www.w3schools.com/cssref/sel_after.asp) | p::after | Insert something after the content of each <p> element |
| [::before](https://www.w3schools.com/cssref/sel_before.asp) | p::before | Insert something before the content of each <p> element |
| [::first-letter](https://www.w3schools.com/cssref/sel_firstletter.asp) | p::first-letter | Selects the first letter of each <p> element |
| [::first-line](https://www.w3schools.com/cssref/sel_firstline.asp) | p::first-line | Selects the first line of each <p> element |
| [::marker](https://www.w3schools.com/cssref/sel_marker.asp) | ::marker | Selects the markers of list items |
| ::selection | p::selection | Selects the portion of an element that is selected by a user |

# How To Add CSS

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

* External CSS
* Internal CSS
* Inline CSS

## **Cascading Order**

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

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

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

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

CSS 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

example:

p.dotted {border-style: dotted;}  
p.dashed {border-style: dashed;}  
p.solid {border-style: solid;}

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

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

p { border: 5px solid red; } 🡪 shorthand

CSS Margins

Margins are used to create space around elements, outside of any defined borders.

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

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

p {  
  margin-top: 25px;  
  margin-bottom: 50px;  
  margin-right: 70px;  
  margin-left: 100px;  
}

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

If the margin property has three values:

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

If the margin property has two values:

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

If the margin property has one value:

* **margin: 25px;**
  + all four margins are 25px
* CSS Padding
* Padding is used to create space around an element's content, inside of any defined borders.

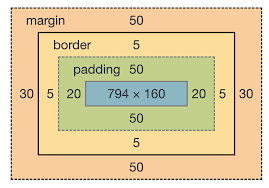
CSS has properties for specifying the padding for each side of an element:

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

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

all shorthand same as margin

# CSS Box Model



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

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

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

CSS Layout - The display Property

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

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

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

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

## **Block-level Elements**

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

Examples of block-level elements:

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

## **Inline Elements**

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

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

Examples of inline elements: <span>, <a>, <img>

|  |  |
| --- | --- |
| inline | Displays an element as an inline element |
| block | Displays an element as a block element |
| contents | Makes the container disappear, making the child elements children of the element the next level up in the DOM |
| flex | Displays an element as a block-level flex container |
| grid | Displays an element as a block-level grid container |
| inline-block | Displays an element as an inline-level block container. The element itself is formatted as an inline element, but you can apply height and width values |
| inline-flex | Displays an element as an inline-level flex container |
| inline-grid | Displays an element as an inline-level grid container |

## **Display: none;**

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

## **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:

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:

## **Override The Default Display Value**

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.

# 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:

## **position: relative;**

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

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

## **position: fixed;**

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

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

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

## **position: absolute;**

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

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

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

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

## **position: sticky;**

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

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

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

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

## **The z-index Property**

When elements are positioned, they can overlap other elements.

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

An element can have a positive or negative stack order:

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

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

## **CSS Overflow**

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

The overflow property has the following values:

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

## **The float Property**

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

The float property can have one of the following values:

* left - The element floats to the left of its container
* right - The element floats to the right of its container
* none - The element does not float (will be displayed just where it occurs in the text). This is default
* inherit - The element inherits the float value of its parent

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

## **The clear Property**

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

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

The clear property can have one of the following values:

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

# CSS 2D Transforms

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

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

* translate()
* rotate()
* scaleX()
* scaleY()
* scale()
* skewX()
* skewY()
* skew()
* matrix()

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

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).

div {  
  transform: scale(2, 3);  
}

## **The skewX() Method**

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

div {  
  transform: skewX(20deg);  
}

# CSS 3D Transforms

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

* rotateX()
* rotateY()
* rotateZ()

CSS Transitions

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

In this chapter you will learn about the following properties:

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

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

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

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

div:hover {  
  width: 300px;  
}

or

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

## **Delay the Transition Effect**

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

The following example has a 1 second delay before starting:

div {  
  transition-delay: 1s;  
}

transition + transformation

div {

width: 100px;

height: 100px;

background: red;

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

}

div:hover {

width: 300px;

height: 300px;

transform: rotate(180deg);

}

# CSS Animations

In this chapter you will learn about the following properties:

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

# 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

<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>

<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);

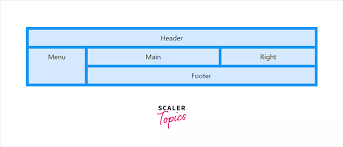
text-align: center;

padding: 20px 0;

font-size: 30px;

}

</style>



# Responsive Web Design - Media Queries

It uses the @media rule to include a block of CSS properties only if a certain condition is true.

# @media only screen and (max-width: 600px) {   body {     background-color: lightblue;   } }

# If the browser window is 600px or smaller, the background color will be lightblue: