1.) 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**. The <div> element is a block-level element. Examples of block-level elements:

<div>

<h1> - <h6>

<p>

<form>

<header>

<footer>

<section>

<li>

Inline Elements

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

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

Examples of inline elements:

<span>

<a>

<img>

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

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

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

li {

display: inline; // By default it is block

}

a {

display: block; // By default, anchor is inline to make it in separate line we have to make it block.

}

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

display:none visibility:hidden Reset

h1.hidden {  
   display: none;  
}

### Difference between display:none and visiblity: hidden

**visibility:hidden** hides the element, but it still takes up space in the layout.

**display:none** removes the element from the document. It does not take up any space.

CSS Display/Visibility Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [display](https://www.w3schools.com/cssref/pr_class_display.asp) | Specifies how an element should be displayed |
| [visibility](https://www.w3schools.com/cssref/pr_class_visibility.asp) | Specifies whether or not an element should be visible |

h2.a {

visibility: visible;

}

h2.b {

visibility: hidden;

}

**Initial**

## Definition and Usage

The initial keyword is used to set a CSS property to its default value.

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

# **CSS inherit Keyword**

* It inherits from the parent CSS.

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

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:

# **CSS Layout - The position Property**

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

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;

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.**

## position: absolute;

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

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

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

Here is a simple example:

This <div> element has position: relative;

This <div> element has position: absolute;

Here is the CSS that is used:

## CSS Box Sizing

The CSS box-sizing property allows us to include the padding and border in an element's total width and height.

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

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. It means you need to worry about only width and the height.**

**body {**

**margin: 0;**

**}**

**\* {**

**box-sizing: border-box;**

**}**

## CSS Box Sizing Property

|  |  |
| --- | --- |
| **Property** | **Description** |
| [box-sizing](https://www.w3schools.com/cssref/css3_pr_box-sizing.asp) | Defines how the width and height of an element are calculated: should they include padding and borders, or not |

## CSS3 Introduced Media Queries

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

* width and height of the viewport
* width and height of the device
* orientation (is the tablet/phone in landscape or portrait mode?)
* resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

## Media Query Syntax

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

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

The result of the query is true if the specified media type matches the type of device the document is being displayed on and all expressions in the media query are true. When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules.

Unless you use the not or only operators, the media type is optional and the all type will be implied.

You can also have different stylesheets for different media:

<link rel="stylesheet" media="mediatype and|not|only (expressions)" href="print.css">

## CSS3 Media Types

|  |  |
| --- | --- |
| **Value** | **Description** |
| all | Used for all media type devices |
| print | Used for printers |
| screen | Used for computer screens, tablets, smart-phones etc. |
| speech | Used for screenreaders that "reads" the page out loud |

One way to use media queries is to have an alternate CSS section right inside your style sheet.

**<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>**

# **Responsive Web Design - Images**

## Using The width Property

If the width property is set to a percentage and the height is set to "auto", the image will be responsive and scale up and down:

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

## Using The max-width Property

If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

### **Example**

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

Different background Images

<https://www.w3schools.com/css/css_rwd_images.asp>

# 1. justify-content: Horizontal

### **Alignment & Spacing along primary axis (X-axis)**

flex-start; Align children horizontally left

flex-end; Align children horizontally right

center; Align children horizontally centered (amaze!)

space-between; Distribute children horizontally evenly across entire width

space-around; Distribute children horizontally evenly across entire width (but with space on the edges

# 2. align-items: Vertical

### **Alignment only along secondary axis (Y-axis)**

flex-start; Align children vertically top

flex-end; Align children vertically bottom

center; Align children vertically centered (amaze!)

baseline; Aligned children vertically so their baselines align (doesn't really work)

stretch; Force children to be height of container (great for columns)