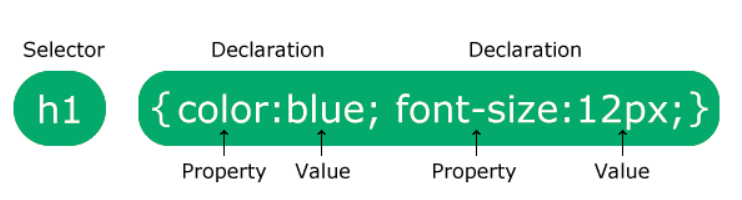
**CSS**

**Css Introduction**

* CSS stands for **Cascading Style Sheets.**
* Used to style HTML document.

**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.
* Declaration blocks are surrounded by curly braces.



* Example:

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

**Explanation**

* p is a selector (it points to the HTML element we 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 comment:**

* Comments are used to explain the code.
* Comment not interpreted by browser.
* Syntax:

/\* comment \*/

**CSS Selectors**

* CSS selectors are used to select the HTML element.
* CSS selectors categories:
* **Simple selectors** (element, id, class)
* **Combinator selectors** (select elements based on 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)

**Simple selectors**

**Element select:**

* Element selector selects HTML elements based on the element name.
* Example:

p {

text-align: center;

color: red;

}

**Id Selector:**

* Id selector uses the **id attribute** of an HTML element to select a specific element.
* 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.
* Note: An id name cannot start with a number.
* Example:

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

**Class Selector:**

* Selects HTML elements with a specific **class attribute** of HTML element.
* To select elements with a specific class, write a period (.) character, followed by the class name.
* In HTML, Multiple element can have same class name.
* And also one element can have multiple class separated by class.
* **Note:** A class name cannot start with a number.
* Example

.center {

text-align: center;

color: red;

}

* We can also specify that only specific HTML elements should be affected by a class.
* Example:

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

/\* Styled applied to only <p> elements with class="center" \*/

**Universal Selector:**

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

\* {

text-align: center;

color: blue;

}

/\* This style apply to all element in html document \*/

**Grouping Selector:**

* It is used to select multiple element with same style.
* Example:

h1, h2, p {

text-align: center;

color: red;

}

/\* This style apply to **h1**, **h2** and **p** element in html document \*/

**Combinator selectors**

**Descendant Selector (space):**

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

div p {

background-color: yellow;

}

/\* selects all <p> elements inside <div> elements \*/

**Direct Child Selector (>):**

* The child selector selects all elements that are the direct children of a specified element.
* Example:

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".
* Example:

div + p {

background-color: yellow;

}

**General Sibling Selector (~)**

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

div ~ p {  
  background-color: yellow;  
}

/\* selects all <p> elements that are **next** siblings of <div> elements \*/

**Pseudo-class selectors:**

* A pseudo-class is used to select a specific state of an element.
* Syntax of pseudo-classes:

selector:pseudo-class {

property: value;

}

**Anchor Pseudo-classes**

* Example:

/\* unvisited link \*/

a:link {

color: #FF0000;

}

/\* visited link \*/

a:visited {

color: #00FF00;

}

/\* mouse over link \*/

a:hover {

color: #FF00FF;

}

/\* selected link \*/

a:active {

color: #0000FF;

}

**Note:**

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

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

**:first-child Pseudo-class**

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

p:first-child {

color: blue;

}

**:lang Pseudo-class**

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

Html:

<p>

Some text

<q lang="fr">A quote in a paragraph</q>

Some text.

</p>

Style:

q:lang(fr) {

background: red;

}

**Pseudo Classes list**

|  |  |  |
| --- | --- | --- |
| [:active](https://www.w3schools.com/cssref/sel_active.asp) | a:active | Selects the active link |
| [:disabled](https://www.w3schools.com/cssref/sel_disabled.asp) | input:disabled | Selects every disabled <input> element |
| [:empty](https://www.w3schools.com/cssref/sel_empty.asp) | p:empty | Selects every <p> element that has no children |
| [:enabled](https://www.w3schools.com/cssref/sel_enabled.asp) | input:enabled | Selects every enabled <input> element |
| [:first-child](https://www.w3schools.com/cssref/sel_firstchild.asp) | p:first-child | Selects every <p> elements that is the first child of its parent |
| [:first-of-type](https://www.w3schools.com/cssref/sel_first-of-type.asp) | p:first-of-type | Selects every <p> element that is the first <p> element of its parent |
| [:focus](https://www.w3schools.com/cssref/sel_focus.asp) | input:focus | Selects the <input> element that has focus |
| [:hover](https://www.w3schools.com/cssref/sel_hover.asp) | a:hover | Selects links on mouse over |
| [:in-range](https://www.w3schools.com/cssref/sel_in-range.asp) | input:in-range | Selects <input> elements with a value within a specified range |
| [:invalid](https://www.w3schools.com/cssref/sel_invalid.asp) | input:invalid | Selects all <input> elements with an invalid value |
| [:lang(*language*)](https://www.w3schools.com/cssref/sel_lang.asp) | p:lang(it) | Selects every <p> element with a lang attribute value starting with "it" |
| [:last-child](https://www.w3schools.com/cssref/sel_last-child.asp) | p:last-child | Selects every <p> elements that is the last child of its parent |
| [:last-of-type](https://www.w3schools.com/cssref/sel_last-of-type.asp) | p:last-of-type | Selects every <p> element that is the last <p> element of its parent |
| [:link](https://www.w3schools.com/cssref/sel_link.asp) | a:link | Selects all unvisited links |
| [:nth-child(n)](https://www.w3schools.com/cssref/sel_nth-child.asp) | p:nth-child(2) | Selects every <p> element that is the second child of its parent |
| [:optional](https://www.w3schools.com/cssref/sel_optional.asp) | input:optional | Selects <input> elements with no "required" attribute |
| [:read-only](https://www.w3schools.com/cssref/sel_read-only.asp) | input:read-only | Selects <input> elements with a "readonly" attribute specified |
| [:required](https://www.w3schools.com/cssref/sel_required.asp) | input:required | Selects <input> elements with a "required" attribute specified |
| [:valid](https://www.w3schools.com/cssref/sel_valid.asp) | input:valid | Selects all <input> elements with a valid value |
| [:visited](https://www.w3schools.com/cssref/sel_visited.asp) | a:visited | Selects all visited links |

**Pseudo-element selectors:**

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

selector::pseudo-element {  
  property: value;  
}

**::first-line Pseudo-element:**

* The ::first-line pseudo-element is used to add a special style to the first line of a text.
* The ::first-line pseudo-element can only be applied to block-level elements.
* Example:

p::first-line {

color: #ff0000;

}

**::first-letter Pseudo-element:**

* The ::first-letter pseudo-element is used to add a special style to the first letter of a text.
* The ::first-letter pseudo-element can only be applied to block-level elements.
* Example:

p::first-letter {

color: #ff0000;

font-size: xx-large;

}

**::before Pseudo-element:**

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

p::before {

content: "Before Psuedo Element";

color: lightblue;

font-size: 2rem;

}

**::after Pseudo-element:**

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

p::after {

content: "After Psuedo Element";

color: lightblue;

font-size: 2rem;

}

**::marker Pseudo-element**

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

::marker {

color: red;

font-size: 23px;

}

**::selection Pseudo-element**

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

::selection {

background-color: lightblue;

color: black;

}

**Attribute selectors:**

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

**[attribute]:**

* Used to select elements with a specified attribute.
* Example:

a[target] {

background-color: yellow;

}

**[attribute=value]:**

* Used to select elements with a specified attribute and value.
* Example:

a[target="\_blank"] {

background-color: yellow;

}

**[attribute~=value]**

* Used to select elements with an attribute value containing a specified word.
* Example:

[title~="flower"] {

border: 5px solid yellow;

}

**[attribute|=value]**

* Used to select elements with the specified attribute, whose value can be exactly the specified value, or the specified value followed by a hyphen (-)
* Example:

[class|="top"] {

background: yellow;

}

**[attribute^=value]**

* Used to select elements with the specified attribute, whose value starts with the specified value.
* Example:

[class^="top"] {

background: yellow;

}

**[attribute$="value"]**

* Used to select elements whose attribute value ends with a specified value.
* Example:

[class$="test"] {

background: yellow;

}

**[attribute\*=value]**

* Used to select elements whose attribute value contains a specified value.
* The value does not have to be a whole word.
* Example:

[class\*="te"] {

background: yellow;

}

**Colors:**

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

**RGB:**

* RGB color value represents Red, Green and Blue
* The value of Red, Green and Blue is between 0 to 255.

rgb(red, green, blue)

**RGBA:**

* RGBA color values are an extension of RGB color values with an alpha channel.
* alpha specifies the opacity for a color.
* The value of alpha if between 0.0 (fully transparent) and 1.0 (not transparent at all).

rgba(red, green, blue, alpha)

**Hex:**

* A hexadecimal color is specified with: #RRGGBB,

where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color.

* RR (red), GG (green) and BB (blue) are hexadecimal values between 00 and ff.

**3 Digit HEX:**

* The 3-digit hex code is a short form for some 6-digit hex codes.
* It specified with: #RGB
* Value of R, G and B is between 0 to f.

**HSL:**

* HSL stands for hue, saturation, and lightness.
* The value of hue is between 0 to 360. 0 is red, 120 is green, and 240 is blue.
* The value of saturation is between 0% to 100%. 0% means a shade of gray, and 100% is the full color.
* The value of lightness is between 0% to 100%. 0% is black and 100% is white.

hsla(hue, saturation, lightness, alpha)

**HSLA:**

* HSLA color values are an extension of HSL color values with an alpha channel
* alpha specifies the opacity for a color.
* The value of alpha is between 0 to 1.

hsla(hue, saturation, lightness, alpha)

**Backgrounds:**

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

**Background color:**

* background-color property specifies the background color of an element.
* Example:

body {

background-color: lightblue;

}

**Opacity:**

* The opacity property specifies the opacity/transparency of an element.
* It can take a value from 0.0 - 1.0.
* Example:

div {

background-color: rgba(0,255,0,0.5);

/\* opacity: 0.3; \*/

}

**Background image:**

* Used to specify the background image of an element.
* Example:

body {

background-image: url("1.png"), url(‘2.png’), url(‘3.png’), url(‘4.png’), linear-gradient(to top right, red, blue);

background-position: top-left, top-right, bottom-left, bottom-right, center;

background-size: 100px, 100px, 100px, 100px, conver;

}

* To specify multiple background image use multiple url(‘’) separated by comma.
* The first image specified by url(‘’), come Top of the other image specified by other url(‘’).
* Example:

.multiple-img {

background-image: url(img\_flwr.gif), url(paper.gif);

background-position: right bottom, left top;

background-repeat: no-repeat, repeat;

padding: 15px;

}

**Background repeat:**

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

: repeat-x;

: repeat-y

: no-repeat

**Background position:**

* The background-position property is used to specify the position of the background image.
* Background-position : x-position y-postion;

**x-position** -> left, right, center, %, px, etc..

**y-postion** -> top, bottom, center, %, px, etc..

**Background attachment:**

* The background-attachment property specifies whether the background image should scroll or be fixed while scrolling the page.
* background-attachment : fixed;

: scroll;

: local;

**Background size:**

* The background-size property specifies the size of the background images.
* background-size : auto

: length

: x-len y-len

: cover

: contain

**Background clip:**

* The background-clip property defines how far the background (color or image) should extend within an element.
* background-clip : border-box

: padding-box

: content-box

**Background origin:**

* The background-origin property specifies the origin position (the background positioning area) of a background image.
* Note: This property has no effect if background-attachment is "fixed" or background-repeat is “repeat”.
* Possible values are content-box, border-box, padding-box.
* Example:

.background-origin {

border: 10px dashed black;

padding: 25px;

background: url(paper.gif);

background-repeat: no-repeat;

background-origin: content-box;

}

**Background Shorthand:**

* It is short handle property for background properties,
* Syntax:

background: bg-color bg-image position bg-size bg-repeat bg-origin bg-clip bg-attachment | initial | inherit;

**Border:**

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

**Border Style:**

* The border-style property specifies what kind of border to display.
* The following values are allowed:

dotted - Defines a dotted border

dashed - Defines a dashed border

solid - Defines a solid border

double - Defines a double border

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

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

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

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

none - Defines no border

hidden - Defines a hidden border

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

* Example:

p.solid {

border-style: solid;

}

p.mix {

border-style: dotted dashed solid double;

}

**Border Width:**

* The border-width property specifies the width of the four borders.
* The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: **thin**, **medium(default)**, or **thick.**
* Example

.border-1 {

border-style: solid;

border-width: 5px;

}

.border2 {

border-style: solid;

border-width: medium;

}

**Border Color:**

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

.border-color {

border-style: solid;

border-color: red;

}

**Border Shorthand Property:**

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

border-width

border-style (required)

border-color

* Example:

p {

border: 1px solid red;

}

**Rounded Borders:**

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

p {

border: 2px solid red;

border-radius: 5px;

}

* **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)
* **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)
* **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)

**Border Image**

**border-image-source:**

* The border-image-source CSS property sets the source image used to create an element's border image.
* Syntax:

border-image-source: url(image.jpg);

border-image-source: linear-gradient(to top, red, yellow);

border-image-source: radial-gradient(red, yellow);

border-image-source: conic-gradient(red, yellow);

**border-image-width:**

* The border-image-width property specifies the width of the border image.
* Syntax

/\* number value \*/

border-image-width: 3;

/\* top and bottom | left and right \*/

border-image-width: 2em 3em;

/\* top | left and right | bottom \*/

border-image-width: 5% 15% 10%;

/\* top | right | bottom | left \*/

border-image-width: 5% 2em 10% auto;

**border-image-slice:**

* The slicing process creates nine regions in total: four **corners**, four **edges**, and a **middle** region.
* Four slice lines, set a given distance from their respective sides, control the size of the regions.
* Syntax:

/\* All sides \*/

border-image-slice: 30%;

/\* top and bottom | left and right \*/

border-image-slice: 10% 30%;

/\* top | left and right | bottom \*/

border-image-slice: 30 30% 45;

/\* top | right | bottom | left \*/

border-image-slice: 7 12 14 5;

**border-image-** **repeat:**

* The border-image-repeat CSS property defines how the edge regions and middle region of a source image are adjusted to fit the dimensions of an element's border image.
* The middle region can be displayed by using the keyword "fill" in the border-image-slice property.
* Syntax:

border-image-repeat: stretch;

border-image-repeat: repeat;

border-image-repeat: round;

border-image-repeat: space;

**border-image-** **outset:**

* The border-image-outset CSS property sets the distance by which an element's border image is set out from its border box.
* Syntax:

/\* <number> value \*/

border-image-outset: 1.5;

/\* top and bottom | left and right \*/

border-image-outset: 1 1.2;

/\* top | left and right | bottom \*/

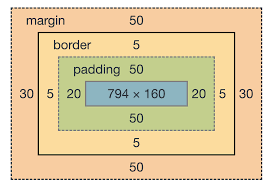
border-image-outset: 30px 2 45px;

/\* top | right | bottom | left \*/

border-image-outset: 7px 12px 14px 5px;

**Box Model:**

* All HTML elements can be considered as boxes.
* The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content.



**Width and Height of an Element**

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

div {

width: 320px;

padding: 10px;

border: 5px solid gray;

margin: 20;

}

Total Width = 320px (content width)

+ 10px + 10px (left + right padding)

+ 5px + 5px (left + right border)

+ 20px + 20px (left + right margin)

= 390px

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

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

**Box Sizing:**

* box-sizing property allows us to include the padding and border in an element's width and height.
* If we set box-sizing: border-box; on an element, padding and border are included in the width and height:

**Total element width** = width + left margin + right margin

**Total element height** = height + top margin + bottom margin

* box-sizing : context-box;

: border-box;

**Margins:**

* Margins are used to create space around elements, outside of any defined borders.
* Properties for specifying the margin for each side of an element:

margin-top

margin-right

margin-bottom

margin-left

* Shorthand Property:

margin: top right bottom left;

margin: top rightleft bottom;

margin: topbottom rightleft;

* Example:

p {

margin-top: 100px;

margin-bottom: 100px;

margin-right: 150px;

margin-left: 80px;

}

* Example:

p {

margin: 25px 50px 75px 100px;

}

* Example:

p {

margin: 25px 50px 75px;

}

**Margins Collapse:**

* Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.
* This does not happen on left and right margins! Only top and bottom margins.
* Example:

h1 {

margin: 0 0 50px 0;

}

h2 {

margin: 20px 0 0 0;

}

**Paddings:**

* Padding is used to create space around an element's content, inside of any defined borders.
* Padding can not have negative value (treated as 0).
* Properties for specifying the padding for each side of an element:

padding-top

padding-right

padding-bottom

padding-left

* Shorthand Property:

padding: top right bottom left;

padding: top rightleft bottom;

padding: topbottom rightleft;

* Example:

div {

padding-top: 50px;

padding-right: 30px;

padding-bottom: 50px;

padding-left: 80px;

}

* Example:

div {

padding: 25px 50px 75px 100px;

}

**Outline:**

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

**Border:**

* A border is used to define the boundaries of an element.
* It is a part of the box model in CSS and can be set using properties like border-width, border-style, and border-color. Borders are typically applied to the edges of an element, and they can be solid, dashed, dotted, etc.
* Borders are an integral part of the layout and design of a webpage and are used to separate and visually distinguish elements.

**Outline:**

* An outline, on the other hand, is similar to a border in that it creates a visual boundary around an element.
* However, outlines are drawn outside the border edge, and they **don't take up space** or affect the layout of the element or surrounding elements. The outline is used mainly for visual emphasis or to indicate focus (such as when navigating a webpage using the keyboard).
* The outline property in CSS allows you to set the style, color, and width of an element's outline.
* It's commonly used in cases where you want to **highlight an element** without affecting its layout. For example, when you click on an input field, it might get an outline to indicate that it is currently focused.
* CSS has the following outline properties:

outline-style

outline-color

outline-width

outline-offset

outline

**Outline style:**

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

dotted - Defines a dotted outline

dashed - Defines a dashed outline

solid - Defines a solid outline

double - Defines a double outline

groove - Defines a 3D grooved outline

ridge - Defines a 3D ridged outline

inset - Defines a 3D inset outline

outset - Defines a 3D outset outline

none - Defines no outline

hidden - Defines a hidden outline

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

**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)(default)

thick (typically 5px)

A specific size (in px, pt, cm, em, etc)

**Outline Color:**

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

p {

outline-style: solid;

outline-color: red;

}

**Outline Offset:**

* The outline-offset property adds space between an outline and the border of an element.
* Example:

p {

margin: 30px;

border: 1px solid black;

outline: 1px solid red;

outline-offset: 15px;

}

**Outline Shorthand:**

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

outline-width

outline-style (required)

outline-color

**Units:**

* CSS has several different units for expressing a length.
* There are two types of length units: absolute and relative.

**Absolute Unit:**

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

|  |  |
| --- | --- |
| cm | Centimetres (1cm = 37.8px) |
| mm | Millimetres (1mm = 3.78px) |
| Q | Quarter millimetres (1Q = ¼mm) |
| In | inches (1in = 96px) |
| px \* | Pixels |
| pt | points (1pt = 1.33px) |
| pc | picas (1pc = 16px) |

**Relative Unit:**

* Relative length units specify a length relative to another length property.

|  |  |
| --- | --- |
| em | Relative to the font-size of the parent element (2em means 2 times the size of the current font of parent). |
| ex | x-height of the parent element's font. |
| ch | Relative to width of the "0" (zero) of parent element. |
| rem | Relative to font-size of the root element. |
| lh | Line height of the parent element. |
| rlh | Line height of the root element. |
| vw | Relative to 1% of the width of the viewport, use ‘visualViewport’ javascript property to check viewpost width and height |
| vh | Relative to 1% of the height of the viewport, use ‘visualViewport’ javascript property to check viewpost width and height |
| vmin | Relative to 1% of viewport's smaller dimension, means smallest of vw and vh. |
| vmax | Relative to 1% of viewport's larger dimension, means largest of vw and vh. |
| % | Relative to the parent element |
| vi | vi = 1% of the size of the viewport’s inline axis. |
| vb | vb = 1% of the size of the viewport’s block axis. |

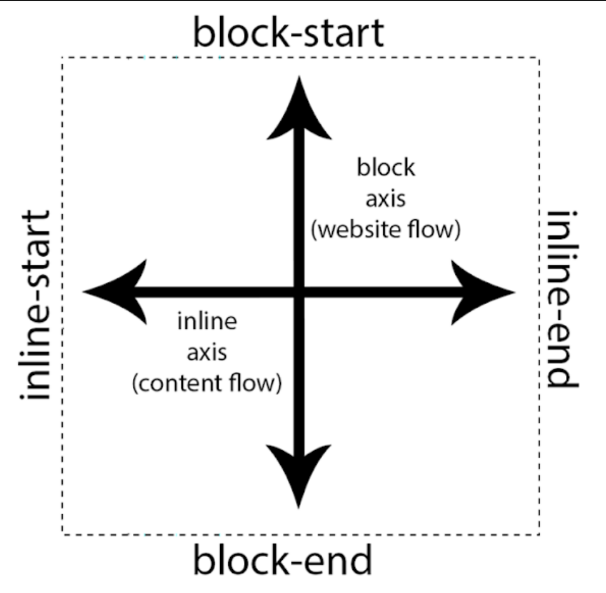
* Viewport is the browser window size.

**x-height:**

* The x-height, or corpus size, is the distance between the baseline and the mean line of lowercase letters in a typeface.

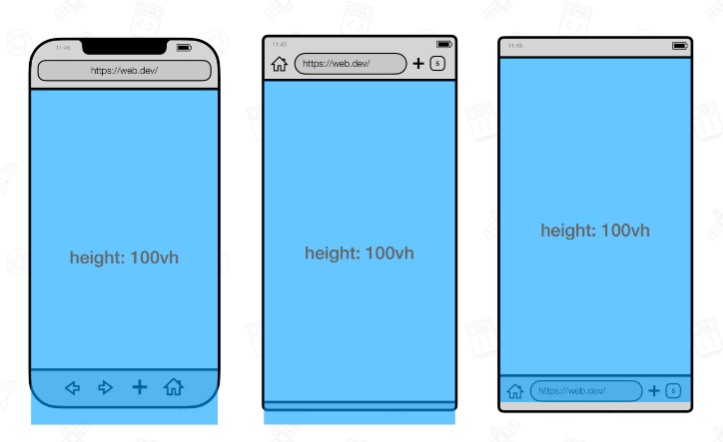


**Inline and Block axis:**

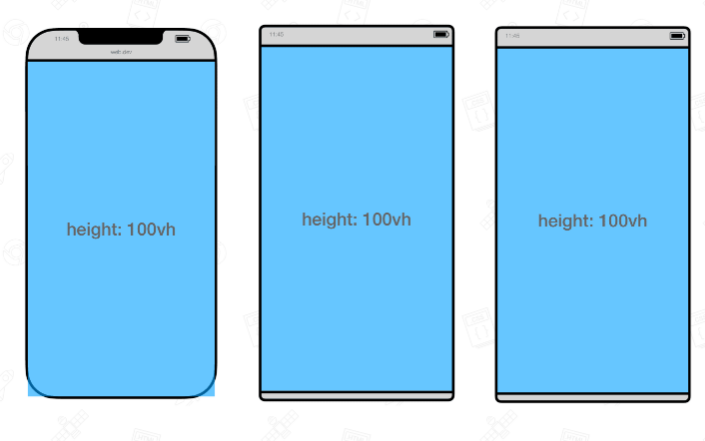
****

**The need for new viewport units:**

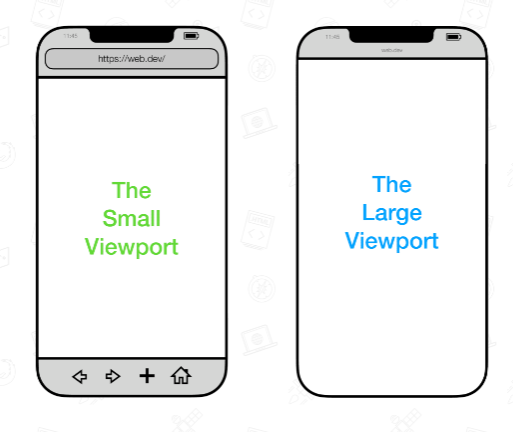
* In Mobile Devices, the viewport size is influenced by the presence or absence of dynamic toolbars. These are user interfaces such as address bars and tab bars.
* Although the viewport size can change, the vw and vh sizes do not. As a result, elements sized to be **100vh** tall will bleed out of the viewport.



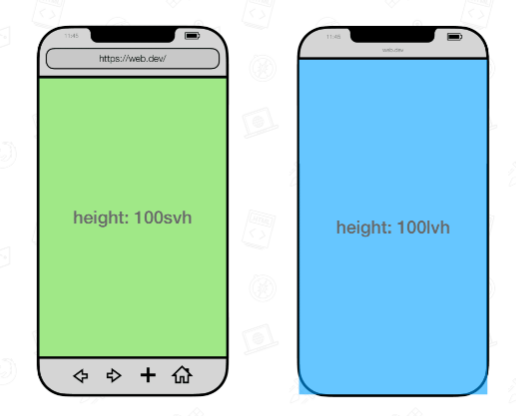
* When scrolling down these dynamic toolbars will retract. In this state, elements sized to be 100vh tall will cover the entire viewport.



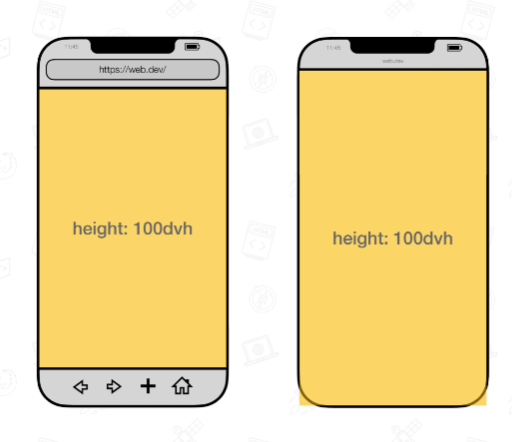
* To solve this problem, the various states of the viewport have been specified at the CSS Working Group.
* **Large viewport:** The viewport sized assuming any U.A. interfaces that are dynamically expanded and retracted to be **retracted**.
* **Small Viewport:** The viewport sized assuming any UA interfaces that are dynamically expanded and retracted to be **expanded**.



* The new viewports also have units assigned to them.
* Units representing the **large viewport** have the **lv** prefix. The units are lvw, lvh, lvi, lvb, lvmin, and lvmax.
* Units representing the **small viewport** have the **sv** prefix. The units are svw, svh, svi, svb, svmin, and svmax.

****

* In addition to the large and small viewports, there‘s also a dynamic viewport which has dynamic consideration of the UA UI:
* When the dynamic toolbars are **expanded**, the dynamic viewport is equal to the size of the **small viewport**.
* When the dynamic toolbars are **retracted**, the dynamic viewport is equal to the size of the **large viewport**.
* Its accompanied units have the **dv** prefix: dvw, dvh, dvi, dvb, dvmin, and dvmax. Their sizes are clamped between their lv\* and sv\* counterparts.



**Text Properties**

**Color property:**

* The color property is used to set the color of the text.
* Example:

h1 {

color: green;

}

**Text alignment property:**

* The text-align property is used to set the horizontal alignment of a text.
* A text can be left or right aligned, cantered, or justified.

left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left.

* Example:

.para-1 {

text-align: center;

}

.para-2 {

text-align: left;

}

.para-3 {

text-align: right;

}

.para-4 {

text-align: justify;

}

**Text align last property:**

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

.para-5 {

text-align-last: right;

}

**Direction property:**

* The direction property in CSS is used to set the base writing direction of text within an element.
* It primarily affects the horizontal text flow, determining whether text is displayed from left to right (LTR) or right to left (RTL).
* Difference between text align and direction property:
* **text-align**: Controls the horizontal alignment of text within an element.
* **direction**: Specifies the base writing direction of text within an element, especially important for languages with **right-to-left scripts**.
* Example:

.para-6 {

direction: rtl;

}

<p class="para-6">Para 6 direction rtl</p>

Output:



**Bidi override property:**

* The unicode-bidi and direction CSS properties work together to determine how bidirectional text is handled in a document.
* The unicode-bidi property specifies which direction the text should run. The direction property sets the default value for text flow.
* Example:

p {

direction: rtl;

unicode-bidi: bidi-override;

}

<p class="para-8">

Para 8 unicode-bidi: bidi-override with direction: rtl;

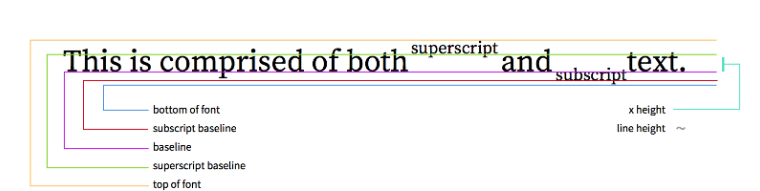
</p>

Output:



**Vertical align property:**

* The vertical-align CSS property sets vertical alignment of an inli**ne, inline-block or table-cell** box.
* The vertical-align property can be used in two contexts:
  + To vertically align an inline-level element's box inside its containing line box. For example, it could be used to vertically position an **image** in a line of text.
  + To vertically align the content of a cell in a table.



* Syntax:

vertical-align: baseline;

vertical-align: sub;

vertical-align: super;

vertical-align: text-top;

vertical-align: text-bottom;

vertical-align: middle;

/\* <length> values \*/

vertical-align: 10em;

vertical-align: 4px;

* Example:

section p img {

width: 10px;

height: 10px;

vertical-align: super;

}

<p>

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Repudiandae, eius ratione aperiam omnis repellat sapiente ut

numquam asperiores recusandae quidem esse dolorem in sed ullam

error ab ipsam qui est! Lorem ipsum dolor sit amet consectetur,

adipisicing elit. Voluptatem consequatur maxime sed accusamus

alias veniam soluta facere eaque deleniti eos. Suscipit pariatur

facilis dolores repudiandae corrupti enim adipisci placeat

natus? <img src="2.png" alt="" />

</p>

Output:



**Text Decoration properties:**

**text-decoration:**

* 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)

* Example:

p {

text-decoration: underline red double 5px;

}

Note: 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.

**text-decoration-line:**

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

.link-1 {

text-decoration-line: overline;

}

.link-2 {

text-decoration-line: line-through;

}

.link-3 {

text-decoration-line: underline;

}

.link-4 {

text-decoration-line: overline underline;

}

**text-decoration-color:**

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

.link-1 {

text-decoration-line: overline;

text-decoration-color: red;

}

**text-decoration-style:**

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

text-decoration-style: solid;

text-decoration-style: double;

text-decoration-style: dotted;

text-decoration-style: dashed;

text-decoration-style: wavy;

* Example:

.wavy {

text-decoration-line: underline;

text-decoration-style: wavy;

text-decoration-color: red;

}

**text-decoration-thickness:**

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

p {

text-decoration-line: underline;

text-decoration-color: red;

text-decoration-style: double;

text-decoration-thickness: 5px;

}

**Text Transform property:**

* It can be used to make text appear in all-uppercase or all-lowercase, or with each word capitalized.
* Example:

text-transform: none;

text-transform: capitalize;

text-transform: uppercase;

text-transform: lowercase;

* Example:

p.uppercase {

text-transform: uppercase;

}

p.lowercase {

text-transform: lowercase;

}

p.capitalize {

text-transform: capitalize;

}

**Text Spacing**

**text-indent:**

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

p {

text-indent: 50px;

}

**letter-spacing:**

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

h1 {

letter-spacing: 5px;

}

**line-height:**

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

p {

line-height: 1.8;

}

/\* default unit is em \*/

**Word Spacing:**

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

p {

word-spacing: 10px;

}

**White Space:**

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

white-space: normal;

white-space: nowrap;

white-space: pre;

white-space: pre-wrap;

white-space: pre-line;

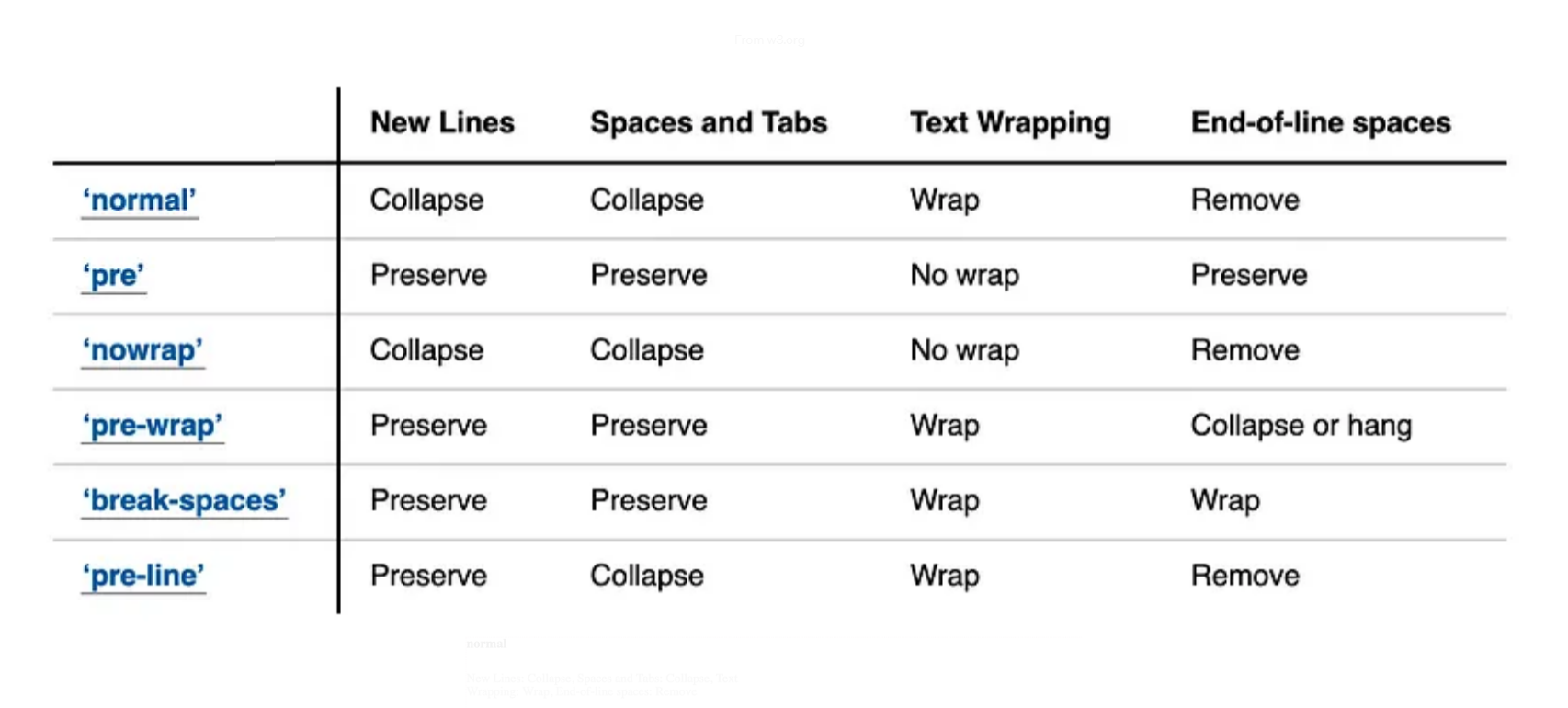
white-space: break-spaces;

* Example:

p {

white-space: nowrap;

}

****

**Solid Text Border:**

* We can use “-webkit-text-stroke” property to outline the text.
* Example:

h1 {

font-size: 40px;

color: transparent;

-webkit-text-stroke: 1px;

}

**Text Overflow:**

* The text-overflow CSS property sets how hidden overflow content is signaled to users. It can be clipped, display an ellipsis ('…'), or display a custom string.
* The text-overflow property doesn't force an overflow to occur. To make text overflow its container, you have to set other CSS properties: overflow and white-space. For example:

overflow: hidden;

white-space: nowrap;

* The text-overflow property only affects content that is overflowing a block container element in its inline progression direction (not text overflowing at the bottom of a box, for example).
* Syntax:

text-overflow: ellipsis;

text-overflow: clip;

**Fonts**

**font-family:**

* The font-family property is used to specify the font of a text.
* Web Safe Font: Web safe fonts are fonts that are universally installed across all browsers and devices.
* Example:

.p1 {

font-family: "Times New Roman", Times, serif;

}

.p2 {

font-family: Arial, Helvetica, sans-serif;

}

.p3 {

font-family: "Lucida Console", "Courier New", monospace;

}

**font-style:**

* The font-style CSS property sets whether a font should be styled with a normal, italic, or oblique face from its font-family.
* Example:

p {

font-style: normal;

}

p {

font-style: italic;

}

p {

font-style: oblique;

}

**font-weight:**

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

font-weight: 100;

font-weight: 200;

font-weight: 300;

font-weight: 400; /\* normal \*/

font-weight: 500;

font-weight: 600;

font-weight: 700; /\* bold \*/

font-weight: 800;

font-weight: 900;

bolder: One relative font weight heavier than the parent element.

Ligher: One relative font weight lighter than the parent element.

* Example:

p.normal {

font-weight: normal;

}

**font-size:**

* The font-size CSS property sets the size of the font. Changing the font size also updates the sizes of the font size-relative <length> units, such as em, ex, and so forth.
* Syntax:

/\* Absolute-size values \*/

/\* Absolute-size keywords, based on the user's default font size of browser (which is medium). \*/

font-size: xx-small;

font-size: x-small;

font-size: small;

font-size: medium;

font-size: large;

font-size: x-large;

font-size: xx-large;

font-size: xxx-large;

/\* Relative-size values \*/

/\* The font will be larger or smaller relative to the parent element's font size \*/

font-size: smaller;

font-size: larger;

/\* length values \*/

font-size: 12px;

font-size: 0.8em;

/\* <percentage> values \*/

font-size: 80%;

**Font-variant:**

* The font-variant property specifies whether or not a text should be displayed in a small-caps font.
* In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.
* Example:

p.normal {

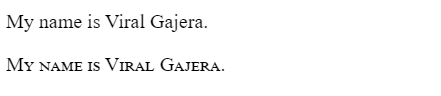
font-variant: normal;

}

p.small {

font-variant: small-caps;

}



**Font property:**

* The font property is a shorthand property for

font-style

font-variant

font-weight

font-size/line-height

font-family

* Example

p {

font: italic small-caps bold 12px/30px Georgia, serif;

}

**Styling List:**

**List-style-type:**

* The list-style-type property specifies the type of list item marker.
* Syntax:

list-style-type: none;

list-style-type: disc;

list-style-type: circle;

list-style-type: square;

list-style-type: decimal;

list-style-type: decimal-leading-zero;

list-style-type: lower-roman;

list-style-type: upper-roman;

list-style-type: upper-alpha;

list-style-type: lower-alpha;

list-style-type: <string>;

* Example:

ul {

list-style-type: circle;

}

ul {

list-style-type: square;

}

**list-style-image:**

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

/\* Keyword values \*/

list-style-image: none;

/\* <url> values \*/

list-style-image: url("starsolid.gif");

/\* valid image values \*/

list-style-image: linear-gradient(to left bottom, red, blue);

* Example:

ul {

list-style-image: url(1.png ');

}

**list-style-position:**

* The list-style-position CSS property sets the position of the ::marker relative to a list item.
* Syntax:

/\* Keyword values \*/

list-style-position: inside;

list-style-position: outside;

* Example:

ol {

list-style-position: inside;

list-style-type: square;

}

**list-style:**

* The list-style CSS shorthand property allows you to set all the list style properties at once.
* This property is a shorthand for the following CSS properties:

list-style-image

list-style-position

list-style-type

**Styling Table**

**Table Borders Property:**

* To specify table borders in CSS, use the border property to table, th, td.
* Example:

table, th, td {

border: 1px solid;

}

**border-collapse:**

* The border-collapse CSS property sets whether cells inside a <table> have shared or separate borders.
* Syntax:

/\* Keyword values \*/

border-collapse: collapse;

border-collapse: separate;

**Table Alignment:**

* The **text-align** property sets the horizontal alignment (like left, right, or center) of the content in <th> or <td>.
* By default, the content of <th> elements are center-aligned and the content of <td> elements are left-aligned.
* The **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).
* Example:

td {

height: 50px;

vertical-align: bottom;

}

**border-spacing:**

* The border-spacing property sets the distance between the borders of adjacent cells.
* Note: This property works only when border-collapse is separate.
* Syntax:

/\* <length> \*/

border-spacing: 2px;

/\* horizontal <length> | vertical <length> \*/

border-spacing: 1cm 2em;

**caption-side:**

* The caption-side [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) property puts the content of a table's [<caption>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/caption) on the specified side.
* Syntax:

/\* Directional values \*/

caption-side: top; (default)

caption-side: bottom;

**empty-cells:**

* The empty-cells property sets whether or not to display **borders** on empty cells in a table.
* Note: This property has no effect if border-collapse is "collapse".
* Syntax:

/\* Keyword values \*/

empty-cells: show;

empty-cells: hide;

**table-layout**:

* The table-layout CSS property sets the algorithm used to lay out <table> cells, rows, and columns.
* Syntax:

/\* Keyword values \*/

table-layout: auto;

table-layout: fixed;

* auto: The automatic table layout algorithm is used. The widths of the table and its cells are adjusted to **fit the content**. Most browsers use this algorithm by default.
* fixed: The fixed table layout algorithm is used. A column element with **explicit width** sets the width for that column.
* Example:

table.a {

table-layout: auto;

width: 180px;

}

table.b {

table-layout: fixed;

width: 180px;

}

**Display property:**

* The display CSS property sets whether an element is treated as a block or inline box and the layout used for its children, such as flow layout (default layout mode), grid or flex.
* The display property sets an element's inner and outer display types. The outer type sets an element's participation in flow layout; the inner type sets the layout of children.
* Syntax:

display: block;

display: inline;

display: inline-block;

display: flex;

display: inline-flex;

/\* Inline-flex: the element behaves like an inline-level element and lays out its content according to the flexbox model. \*/

display: grid;

display: inline-grid;

/\* The element behaves like an inline-level element and lays out its content according to the grid model. \*/

display: none;

display: contents;

/\* In CSS, the display: contents property makes an element's children appear as if they were direct children of the element's parent. This essentially makes the container disappear. \*/

display: inline flow-root; /\* similar to inline-block \*/

display: block flex; /\* flex \*/

display: inline flex; /\* inline-flex \*/

display: block grid; /\* grid \*/

display: inline grid; /\* inline-grid \*/

display: block flow-root; /\* inline-block \*/

display: table;

/\* These elements behave like HTML table elements. It defines a block-level box \*/

display: table-row;

/\* These elements behave like <tr>, table elements. It defines a block-level box \*/

/\* all table elements have an equivalent CSS display value \*/

display: list-item;

/\* The element generates a block box for the content and a separate list-item inline box. \*/

**Visibility:**

* The visibility CSS property shows or hides an element without changing the layout of a document. The property can also hide rows or columns in a <table>.
* Syntax:

visibility: visible;

visibility: hidden;

/\* The element box is invisible (not drawn), but still affects layout as normal. \*/

visibility: collapse;

/\* For <table> rows, columns, column groups, and row groups, the row(s) or column(s) are hidden and the space they would have occupied is removed (as if display: none were applied to the column/row of the table). \*/

/\* For other elements, collapse is treated the same as hidden. \*/

**Position Property:**

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

position: static;

/\* This is the default value. The element is positioned according to the normal flow of the document. The top, right, bottom, left, and z-index properties have no effect. \*/

position: relative;

/\* The element is positioned according to the normal flow of the document, and then offset relative to itself based on the values of top, right, bottom, and left. \*/

position: absolute;

/\* The element is removed from the normal document flow, and no space is created for the element in the page layout. \*/

/\* The element is positioned relative to its closest positioned ancestor (if any) or to the initial containing block. \*/

position: fixed;

/\* The element is removed from the normal document flow, and no space is created for the element in the page layout. \*/

/\* The element is positioned relative to its initial containing block, which is the viewport in the case of visual media. \*/

/\* Its final position is determined by the values of top, right, bottom, and left. \*/

position: sticky;

/\* The element is positioned according to the normal flow of the document, and then offset relative to its nearest scrolling ancestor and containing block \*/

/\* A sticky element toggle 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 ‘sticky’ in place (like position fixed) \*/

**z-index Property:**

* The z-index CSS property sets the z-order of a “positioned” element and its descendants or “flex” and “grid” items. Overlapping elements with a larger z-index cover those with a smaller one.
* Syntax:

z-index: auto;  
/\* This is default \*/

z-index: 1;

z-index: 3;

z-index: 289;

z-index: -1;

/\* Negative values to lower the priority \*/

**Overflow:**

**overflow-x:**

* The overflow-x CSS property sets what shows when content overflows a block-level element's left and right edges.
* Syntax:

overflow-x: visible; (default)

overflow-x: hidden;

/\* Overflow content is clipped if necessary to fit horizontally in the elements' padding box. \*/

overflow-x: clip;

/\* Overflow content is clipped at the element's overflow clip edge that is defined using the overflow-clip-margin property. \*/

overflow-x: scroll;

overflow-x: auto;

**overflow-y:**

* The overflow-y CSS property sets what shows when content overflows a block-level element's top and bottom edges.
* Syntax:

overflow-y: visible; (default)

overflow-y: hidden;

overflow-y: clip;

overflow-y: scroll;

overflow-y: auto;

**overflow:**

* The overflow CSS shorthand property sets the desired behavior when content does not fit in the parent element box (overflows) in the horizontal and/or vertical direction.
* Syntax:

overflow: visible;

overflow: hidden;

overflow: clip;

overflow: scroll;

overflow: auto;

**overflow-clip-margin:**

* The overflow-clip-margin CSS property determines how far outside its bounds an element with overflow: clip may be painted before being clipped.

**Float Property:**

**Float:**

* The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it.
* Syntax:

float: left;

float: right;

float: none;

float: inline-start;

/\* It Floats the left side with ltr scripts, and the right side with rtl scripts.\*/

float: inline-end;

/\* It Floats the right side with ltr scripts, and the left side with rtl scripts. \*/

**Clear:**

* The clear CSS property sets whether an element must be moved below (cleared) floating elements that precede it.
* It moves the border edge of the element down until it is below the margin edge of all relevant floats. The non-floated block's top margin collapses.
* Vertical margins between two floated elements on the other hand will not collapse.
* Syntax:

clear: none;

clear: left;

clear: right;

clear: both;

clear: inline-start;

/\* inline-start keyword indicating that the element is moved down to clear floats on start side of its containing block, that is the left floats on ltr scripts and the right floats on rtl scripts. \*/

clear: inline-end;

/\* inline-end keyword indicating that the element is moved down to clear floats on end side of its containing block, that is the right floats on ltr scripts and the left floats on rtl scripts. \*/

**!important Rule:**

* The !important rule in CSS is used to add more importance to a property/value than normal.
* If we use the !important rule, it will override ALL previous styling rules for that specific property on that element!
* Example:

p {

background-color: red !important;

}

**Shadow Property:**

* We can add shadow to text and elements, using following properties…
  + 1. text-shadow
    2. box-shadow

**text-shadow:**

* The text-shadow CSS property adds shadows to text. It accepts a comma-separated list of shadows to be applied to the text and any of its text-decorations.
* Syntax:

text-shadow: x-offset y-offset blur colour;

* Example:

h1 {

text-shadow: 0 0 3px #FF0000;

}

* To add more than one shadow to the text, you can add a comma-separated list of shadows.
* Example:

h1 {

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

}

**box-shadow:**

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

box-shadow: x-offset y-offset blur spread color inset;

* Example:

div {

box-shadow: 10px 10px 5px 12px lightblue;

}

* To add more than one shadow to the text, you can add a comma-separated list of shadows.
* Example:

div {

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

}

**2D Transforms:**

* The transform [CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) property lets you rotate, scale, skew, translate etc.. an element.
* Only transformable elements can be transformed. That is, all elements whose layout is governed by the CSS box model except for: inline boxes, table-column boxes, and table-column-group boxes.
* Syntax:

/\* Keyword values \*/

transform: none;

/\* Function values \*/

transform: translate(12px, 50%);

transform: translateX(2em);

transform: translateY(3in);

transform: rotate(0.5turn);

transform: rotateX(10deg);

transform: rotateY(10deg);

transform: scale(2, 0.5);

transform: scaleX(2);

transform: scaleY(0.5);

transform: skew(30deg, 20deg);

transform: skewX(30deg);

transform: skewY(1.07rad);

transform: perspective(17px);

/\* Multiple function values \*/

transform: translateX(10px) rotate(10deg) translateY(5px);

transform: perspective(500px) translate(10px, 0, 20px) rotateY(3deg);

**none:**

* Specifies that no transform should be applied.

**translate():**

* The translate() CSS function repositions an element in the horizontal and/or vertical directions.
* Syntax:

transform: translate(x-pos)

transform: translate(x-pos, y-pos)

Note: the percentage value is relative to element dimension (i.e. element’s width and height), not relative to parent’s width and height.

translateX(): Translates an element horizontally.

translateY(): Translates an element vertically.

* Example:

.translate-1 {

width: 100px;

height: 100px;

background-color: red;

transform: translate(100px, 20px);

}

.translate-2 {

width: 100px;

height: 100px;

background-color: blue;

transform: translate(100%);

}

**rotate():**

* The rotate(angle) CSS function defines a transformation that rotates an element around a fixed point on the 2D plane, without deforming it.
* Practically it rotate around Z-axis.
* “angle" represents the angle of the rotation. The direction of rotation depends on the **writing direction**. In a left-to-right context, a positive angle denotes a clockwise rotation, a negative angle a counter-clockwise one. In a right-to-left context, a positive angle denotes a counter-clockwise rotation, a negative angle a clockwise one.
* RotateX(angle): The rotateX function defines a transformation that rotates an element around the x-axis (horizontal) without deforming it.
* RotateY(angle): The rotateY function defines a transformation that rotates an element around the y-axis (vertical) without deforming it.
* RotateZ(angle): The rotate function defines a transformation that rotates an element around the z-axis without deforming it.
* Example:

.rotate {

width: 100px;

height: 100px;

background-color: aqua;

transform: rotate(45deg);

}

.rotate-x {

width: 100px;

height: 100px;

background-color: red;

transform: rotateX(45deg);

}

.rotate-y {

width: 100px;

height: 100px;

background-color: rebeccapurple;

transform: rotateY(45deg);

}

.rotate-z {

width: 100px;

height: 100px;

background-color: brown;

transform: rotateZ(45deg);

}

**scale():**

* The scale() function defines a transformation that resizes an element on the 2D plane. Because the amount of scaling is defined by a x and y, it can resize the horizontal and vertical dimensions at different scales.
* scale(x)
* scale(x,y)
* scaleX(x)
* scaleY(y)
* Only Percentage and number values are allowed (number value without units).
* Example:

.scale-1 {

width: 100px;

height: 100px;

background-color: antiquewhite;

transform: scale(1.5);

}

.scale-2 {

width: 100px;

height: 100px;

background-color: antiquewhite;

transform: scale(2, 1.2);

}

.scale-x {

width: 100px;

height: 100px;

background-color: antiquewhite;

transform: scaleX(1.5);

}

.scale-y {

width: 100px;

height: 100px;

background-color: antiquewhite;

transform: scaleY(150%);

/\* transform: scaleY(1.5); \*/

}

**skew():**

* The skew() CSS function defines a transformation that skews an element on the 2D plane.
* skew(x)
* skew(x,y)
* skewX(x)
* skewY(y)
* The effect is as if you grabbed each corner of the element and pulled them along a certain angle.
* Allowed values is any valid angle(deg, rad, trun)…
* Example:

.skew-1 {

width: 100px;

height: 100px;

background-color: rosybrown;

transform: skew(45deg);

}

.skew-2 {

width: 100px;

height: 100px;

background-color: aqua;

transform: skew(0deg, 45deg);

}

.skew-x {

width: 100px;

height: 100px;

background-color: red;

transform: skew(45deg);

}

.skew-y {

width: 100px;

height: 100px;

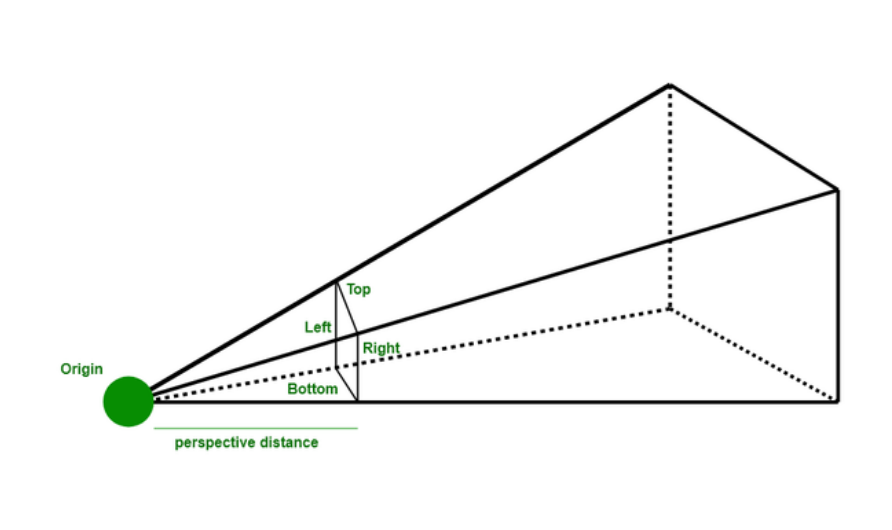
background-color: bisque;

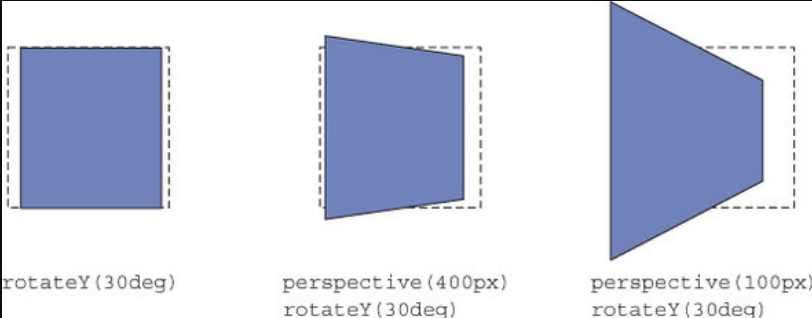
transform: skew(0deg, 45deg);

}

**perspective():**

* The perspective(distance) function defines a transformation that sets the distance between the user and the z=0 plane (XY plane is Z=0 plane).
* The perspective distance used by perspective() is specified by a length(absolute and relative) value.
* The z=0 plane is the plane where everything appears in a 2-dimensional view, or the screen.
* Negative values are syntax errors.
* Values smaller than 1px (including zero) are clamped to 1px.
* Large values of perspective represent a small transformation; small values of perspective() represent a large transformation;
* perspective(none) represents perspective from infinite distance and no transformation.





**Transitions:**

* CSS transitions allows you to change property values smoothly, over a given duration.
* Transitions enable you to define the transition between two states of an element. Different states may be defined using pseudo-classes like: hover or: active or dynamically set using JavaScript.

**transition-property:**

* The transition-property used to set the CSS properties to which a transition effect should be applied.
* not all property values can be used as values for the transition-property CSS property.
* Transition-property can accept only valid CSS property names, such as color, background-color, and transform.
* Other property names (such as flex, grid, and display) are not compatible with this property.
* Syntax:

/\* Keyword values \*/

transition-property: none;

transition-property: all; /\* default \*/

/\* values as property name \*/

transition-property: color;

transition-property: height, color;

**transition-duration:**

* The transition-duration CSS property sets the length of time a transition animation should take to complete.
* By default, the value is **0s**, meaning that no animation will occur.
* You may specify multiple durations; each duration will be applied to the corresponding property as specified by the transition-property property,
* A negative value for the time renders the declaration invalid.
* Syntax:

/\* <time> values \*/

transition-duration: 6s;

transition-duration: 120ms;

transition-duration: 1s, 15s;

transition-duration: 10s, 30s, 230ms;

**transition-timing-function:**

* The transition-timing-function property specifies the speed curve of the transition effect.
* The transition-timing-function property can have the following values:
* ease - specifies a transition effect with a slow start, then fast, then end slowly (this is default)
* linear - specifies a transition effect with the same speed from start to end
* ease-in - specifies a transition effect with a slow start
* ease-out - specifies a transition effect with a slow end
* ease-in-out - specifies a transition effect with a slow start and end
* cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier function

/\* Keyword values \*/

transition-timing-function: ease; /\*default\*/

transition-timing-function: ease-in;

transition-timing-function: ease-out;

transition-timing-function: ease-in-out;

transition-timing-function: linear;

transition-timing-function: step-start;

transition-timing-function: step-end;

/\* Function values \*/

transition-timing-function: cubic-bezier(0.1, 0.7, 1, 0.1);

transition-timing-function: steps(4, jump-end);

/\* Steps Function keywords \*/

transition-timing-function: steps(4, jump-start);

transition-timing-function: steps(10, jump-end);

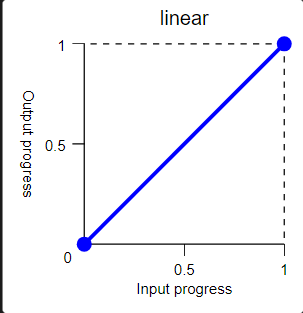
transition-timing-function: steps(20, jump-none);

transition-timing-function: steps(5, jump-both);

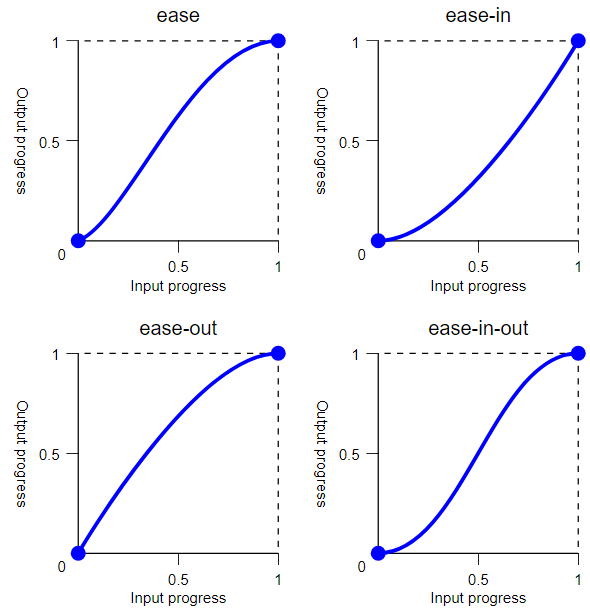
transition-timing-function: steps(6, start);

transition-timing-function: steps(8, end);

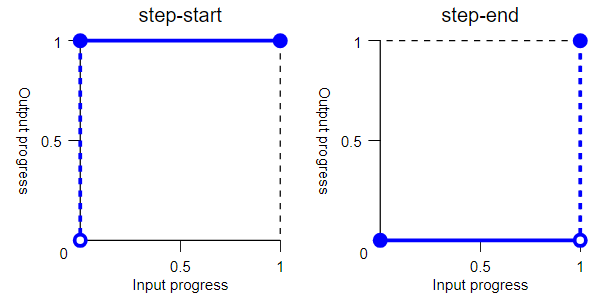
**Linear:**

****

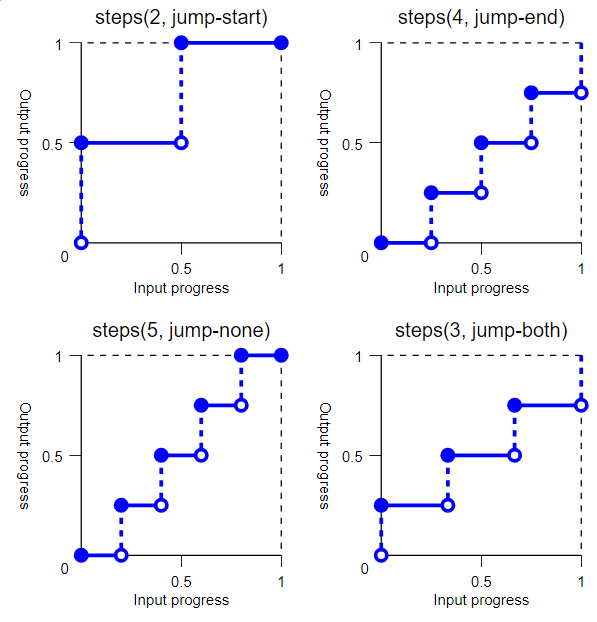
**Ease, ease-in, ease-out, ease-in-out:**

****

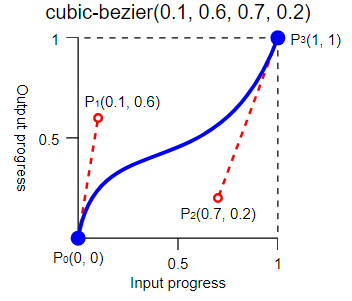
**Step-start, step-end:**

****

**Step-function:**

****

**cubic-bezier:**

****

**transition-delay:**

* The transition-delay CSS property specifies the duration to wait before starting a property's transition effect when its value changes.
* Syntax:

/\* <time> values \*/

transition-delay: 3s;

transition-delay: 2s, 4ms;

**transition:**

* The transition CSS property is a shorthand property for transition-property, transition-duration, transition-timing-function, and transition-delay.
* Syntax:

/\* property name | duration \*/

transition: margin-right 4s;

/\* property name | duration | delay \*/

transition: margin-right 4s 1s;

/\* property name | duration | easing function \*/

transition: margin-right 4s ease-in-out;

/\* property name | duration | easing function | delay \*/

transition: margin-right 4s ease-in-out 1s;

* Example (transition property):

.container {

border: 1px solid black;

margin: 10px;

padding: 10px;

}

.box {

/\* border: 1px solid black; \*/

margin: 10px;

padding: 10px;

width: 200px;

}

.box1 {

background-color: violet;

/\* transition: all; \*/

transition: width 2s;

}

.box1:hover {

width: 95%;

background-color: red;

}

.box2 {

background-color: violet;

/\* transition: all; \*/

transition: all 2s;

}

.box2:hover {

width: 95%;

background-color: red;

}

* Example (transition timing function):

.container {

border: 1px solid black;

margin: 10px;

padding: 10px;

}

.box {

/\* border: 1px solid gainsboro; \*/

margin: 10px;

padding: 10px;

width: 200px;

}

.box1 {

background-color: violet;

transition: all 2s ease;

}

.box2 {

background-color: aqua;

transition: all 2s ease-in;

}

.box3 {

background-color: yellow;

transition: all 2s ease-out;

}

.box4 {

background-color: pink;

transition: all 2s ease-in-out;

}

.box5 {

background-color: lightblue;

transition: all 2s linear;

}

.box6 {

background-color: lightcoral;

transition: all 2s step-start;

}

.box7 {

background-color: lightgreen;

transition: all 2s step-end;

}

.container:hover .box {

width: 95%;

}

* Example (transition delay):

.container {

border: 1px solid black;

margin: 10px;

padding: 10px;

}

.box {

margin: 10px;

padding: 10px;

width: 100px;

}

.box1 {

background-color: lightseagreen;

transition: all 1s 1s;

}

.box2 {

background-color: lightcoral;

transition: all 1s 2s;

}

.box1:hover {

width: 95%;

}

.container:hover .box {

width: 95%;

}

**Media Queries**

* Media queries allow you to apply CSS styles depending on a device's general **type** (such as print vs. screen) or other characteristics such as **screen resolution** or browser viewport **width**.

**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 rule\*/

}

* 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.

You can also have different stylesheets for different media:

* Syntax:

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

* Example:

<head>

<title>Document</title>

<link rel="stylesheet" href="style-1.css" />

<link rel="stylesheet" href="style-2.css" media="all and (max-width:768px)" />

</head>

<body>

<h1>Hello World</h1>

</body>

**Media Types:**

* Media types describe the general category of a device. Media type can be any of screen, print, speech or **all(default)**.
* The Media type is optional.
* Printing in browsers usually ignores background colors.
* Example:

.container {

width: 100px;

height: 100px;

border: 10px solid black;

}

@media print {

.container {

border: 10px solid red;

}

}

**Logical Operators:**

* The logical operators **not**, **and**, **only**, and **or** can be used to compose a complex media query.
* We can also combine multiple media queries into a single rule by separating them with **commas**(,).
* Example:

.container {

width: 100px;

height: 100px;

border: 10px solid black;

}

@media not print {

.container {

border: 10px solid red;

}

}

**Media features:**

* Media features describe specific characteristics of the user agent/output device, or environment.
* Media feature expressions test for their presence or value, and are entirely optional.
* Each media feature expression must be surrounded by parentheses.

width

min-width

max-width

height

min-height

max-height

any-hover (hover|none) /\* hover effect should be apply or not\*/

aspect-ration (width/height) /\* checks if width/height ration ratio matches \*/

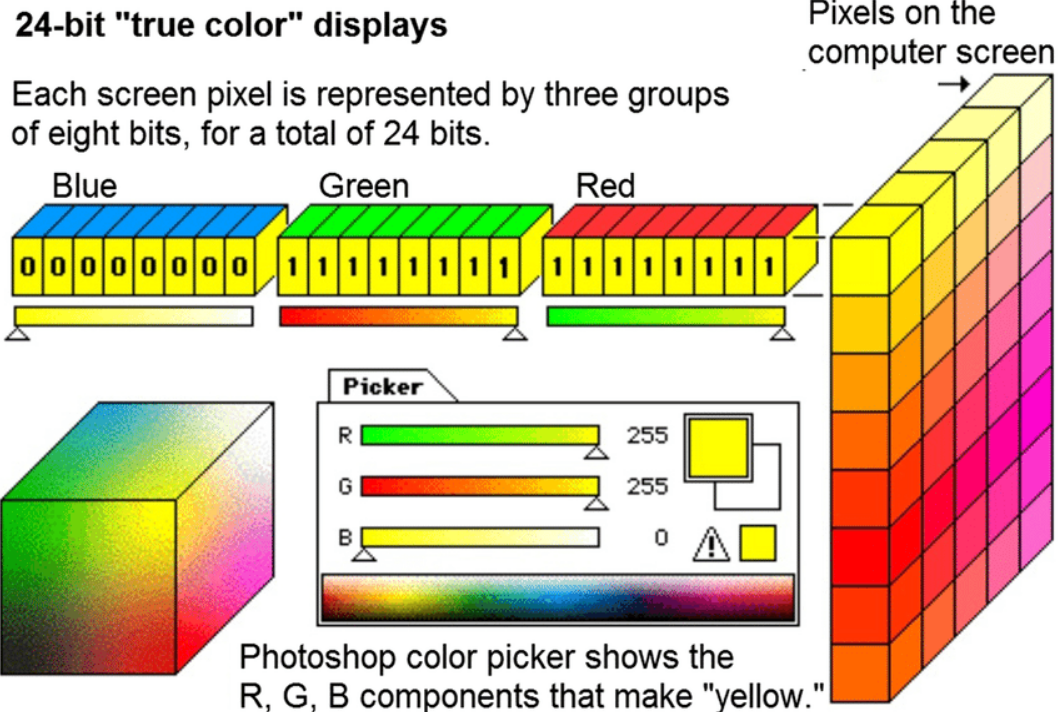
color /\* If the device is not a color device, the value is zero.

Also used to test the number of bits per color component

\*/

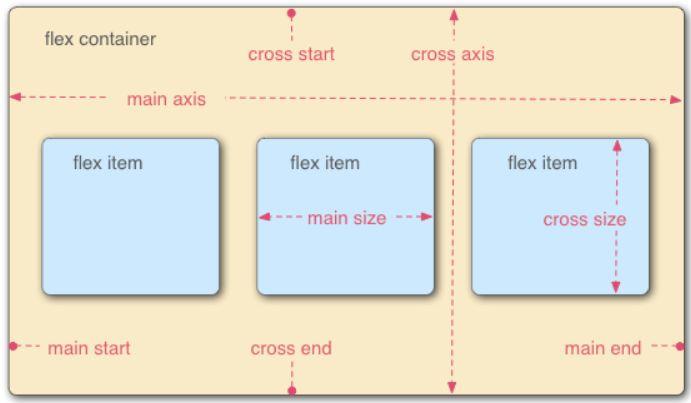
orientation /\* test the orientation of the viewport, landscape or portrait \*/

prefers-color-scheme



**Flex Box**

* Flexbox is a one-dimensional layout method for arranging items in rows or columns.
* The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.
* To start using the Flexbox model, you need to first define a flex container, using **display: flex;** property.



**Flex Container:**

* The element becomes flexible(flex) by setting the display property to flex.
* Flex container contains one or more flex items.
* The flex container properties are:
* flex-direction
* flex-wrap
* flex-flow
* justify-content
* align-items
* align-content

**flex direction:**

* The flex-direction CSS property sets how flex items are placed in the flex container defining the main axis and the direction (normal or reversed).
* By default, this is set to row, which causes them to be laid out in a row in the direction your browser's default language works in (left to right, in the case of an English browser).
* Syntax:

flex-direction: row;

flex-direction: row-reverse;

flex-direction: column;

flex-direction: column-reverse;

* Example:

.container {

border: 1px solid black;

margin: 10px;

padding: 10px;

display: flex;

flex-direction: row;

/\* flex-direction: row-reverse; \*/

/\* flex-direction: column; \*/

/\* flex-direction: column-reverse; \*/

}

.item {

border: 1px solid black;

width: 100px;

height: 100px;

margin: 10px;

text-align: center;

font-size: xx-large;

}

**flex wrap:**

* The flex-wrap CSS property sets whether flex items are forced onto one line or can wrap onto multiple lines.
* Syntax:

flex-wrap: nowrap; /\* Default value \*/

flex-wrap: wrap;

flex-wrap: wrap-reverse; /\* cross-start and cross-end are permuted \*/

* Example:

.container {

border: 1px solid black;

margin: 10px;

padding: 10px;

display: flex;

flex-direction: row;

/\* flex-wrap: nowrap; \*/

/\* flex-wrap: wrap; \*/

flex-wrap: wrap-reverse;

}

.item {

border: 1px solid black;

width: 100px;

height: 100px;

margin: 10px;

text-align: center;

font-size: xx-large;

}

**Justify-content Property:**

* justify-content controls where the flex items sit on the main axis (horizontal axis).
* Syntax:

justify-content: center; /\* Pack items around the center \*/

justify-content: start; /\* Pack items from the start \*/

justify-content: end; /\* Pack items from the end \*/

/\* **flex-start** takes into account the presence of the -reverse values of the flex direction, while **start** does not. \*/

/\* **start** and **flex-start** Depends on the page or DOM direction (RTL, LTR), if the page is ltr then **start** refers to left and end refers to right \*/

justify-content: flex-start; /\* Pack flex items from the start \*/

justify-content: flex-end; /\* Pack flex items from the end \*/

justify-content: left; /\* Pack items from the left \*/

justify-content: right; /\* Pack items from the right \*/

/\* Normal alignment (default) \*/

justify-content: normal;

/\* Distributed alignment \*/

justify-content: space-between; /\* Distribute items evenly

The first item is flush with the start,

the last is flush with the end \*/

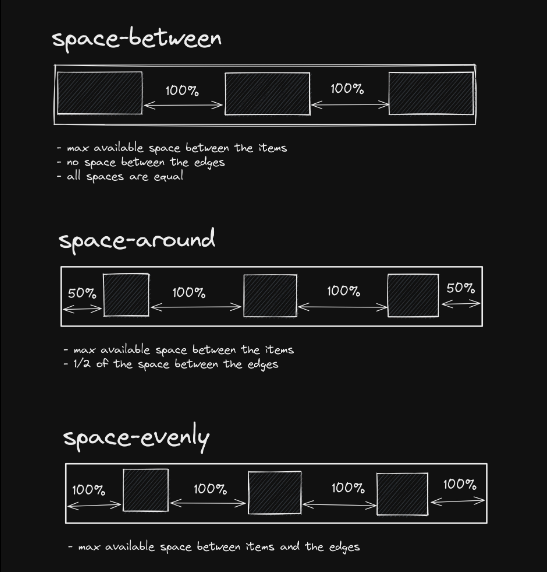
justify-content: space-around; /\* Distribute items evenly

Start and end gaps are half the size of the space

between each item \*/

justify-content: space-evenly; /\* Distribute items evenly

Start, in-between, and end gaps have equal sizes \*/



**align-items:**

* In Flexbox, it controls the alignment of items on the Cross Axis (Vertical Alignment).
* The CSS align-items property sets the align-self value on all direct children as a group.
* Syntax:

/\* Basic keywords \*/

align-items: normal;

/\* Positional alignment \*/

/\* align-items does not take left and right values \*/

align-items: center;

align-items: start;

align-items: end;

align-items: flex-start;

align-items: flex-end;

align-items: self-start;

align-items: self-end;

/\* Baseline alignment \*/

align-items: baseline;

align-items: first baseline;

align-items: last baseline;