

CSS

CSS

- **CSS** stands for **Cascading Style Sheets**
 - Styles define **how to display** HTML elements
 - Styles are normally stored in **Style Sheets**
 - Multiple style definitions will **cascade** into one

Anatomy of CSS Rule

Syntax

- Is made up of three parts:
 - a selector, a property and a value:

```
selector {  
  property: value;  
}
```

```
• Ex: body {  
  color: black;  
}
```

Element selector

- If the value is multiple words, put quotes around the value:
 - `p {
font-family: "Times New Roman", Times, serif;
}`
- To specify more than one property, you must separate each property with a semicolon.
 - `p {
text-align:center; color:red
}`

Class selector

- With the class selector, different styles can be defined for the same type of HTML element.
 - `p.right {text-align: right}`
 - `p.center {text-align: center}`
 - `p.bold{font-weight:bold}`
- You have to use the class attribute in your HTML element:
 - `<p class="right"> This paragraph will be right-aligned. </p>`
 - `<p class="center"> This paragraph will be center-aligned. </p>`
- To apply more than one class per given element, the syntax is:
 - `<p class="center bold"> This is a paragraph. </p>`
 - The paragraph above will be styled by the class "center" AND the class "bold".

Anonymous class selector

- The tag name in the selector can be omitted to define a style that will be used by all HTML elements that have a certain class.
- In the example below, all HTML elements with class="center" will be center-aligned:
- `.center {text-align: center;}`
- Ex: `<h1 class="center"> This heading will be center-aligned </h1><p class="center"> This paragraph will also be center-aligned. </p>`

Add Styles to Elements with Particular Attributes

- Styles can be applied to HTML elements with particular attributes.
- The style rule below will match all input elements that have a type attribute with a value of "text":
- Ex: `input[type="text"] {background-color: blue}`

ID selector

- Define styles for HTML elements with the id selector.
- The id selector is defined as a #.
- The style rule below can be used with an element that has an id with a value of "para1":
- `#para1 { text-align: center; color: red }`

External Style Sheet

```
<head>
```

```
<link rel="StyleSheet" href="mystyle.css" type="text/css" />
```

```
</head>
```

Internal Style Sheet

```
<head>
<style type="text/css">
  h1 {color: red}
  p {margin-left: 20px}
  h1.BlueOnes{color:blue;}
  h1.RedOnes{color:red;}
  #hcolor{color:red;}
</style>
</head>
```

```
<h1>Am I in red</h1>
<h1 CLASS =“BlueOnes” >Blue color text</h1>
<h1 id=“hcolor”>Red color text</h1>
```

Inline Styles

`<p style="color: sienna; margin-left: 20px"> This is a paragraph </p>`

CASCADING effect

- Styles can be specified
 - inside a single HTML element
 - inside the <head> element of an HTML page
 - In an external CSS file.
- Even multiple external style sheets can be referenced inside a single HTML document.
- All the styles will "cascade" into a new "virtual" style sheet by the priority rules:
 - Browser default
 - External style sheet or Linked
 - Internal style sheet or Embedded (inside the <head> tag)
 - Inline style (inside an HTML element)

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="StyleSheet" href="../CSS/mystyle.css"
type="text/css" />
<style>
body{ background-color:black
}
h1 {
border:1px orange inset;
color: white;
text-align: center
}
</style>
</head>
```

```
<body>
<h1 style="color:yellow">My First CSS
Example</h1>
</body>
</html>
```

```
//mystyle.css
h1 {
color: red;
text-align: right;
height:100px;
/* background-color:gray; */
}
```

Box-model



Margin

- margin: *length* | auto | initial | inherit;
- Default: 0
- margin: 10px 5px 15px 20px;
 - top margin is 10px
 - right margin is 5px
 - bottom margin is 15px
 - left margin is 20px
- margin: 10px 5px 15px;
 - top margin is 10px
 - right and left margins are 5px
 - bottom margin is 15px
- margin: 10px 5px;
 - top and bottom margins are 10px
 - right and left margins are 5px
- margin: 10px;
 - all four margins are 10px

Border

- border: *border-width border-style border-color* | initial | inherit;
- Default: medium none elementcolor
- border: 3px inset red;

Eg: border-bottom: 6px solid red;
background-color: lightgrey;
border-color: red green blue yellow;
border-top-style: dotted;

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

Border radius

- border-radius: *1-4 length | % / 1-4 length | %* | initial | inherit;
- Default: 0
- border-radius: 15px 50px 30px 5px:
- border-radius: 15px 50px 30px:
- border-radius: 15px 50px:
- border-top-left-radius: 1em 5em
border-top-right-radius
border-bottom-right-radius
border-bottom-left-radius

Padding

- padding: *length* | initial | inherit;
- padding: 2cm 4cm 3cm 4cm;

Dimension

- height: auto | *length* | initial | inherit;
- width: auto | *value* | initial | inherit;
- max-height: none | *length* | initial | inherit;
- max-width: none | *length* | initial | inherit;
- min-height: *length* | initial | inherit;
- min-width: *length* | initial | inherit;

Overflow

- **Overflow:** Specifies what happens if content overflows an element's box
- **overflow-x:** Specifies what to do with the left/right edges of the content if it overflows the element's content area
- **overflow-y:** Specifies what to do with the top/bottom edges of the content if it overflows the element's content area
 - (visible, hidden, auto, scroll)

Box-sizing

- By default, the width and height of an element is calculated like this:
 - $\text{width} + \text{padding} + \text{border} = \text{actual width of an element}$
 - $\text{height} + \text{padding} + \text{border} = \text{actual height of an element}$
- The CSS box-sizing property allows us to include the padding and border in an element's total width and height.
- **box-sizing: content-box | border-box | initial | inherit;**
 - **content-box:** Default. The width and height properties (and min/max properties) includes only the content. Border and padding are not included
 - **border-box:** The width and height properties (and min/max properties) includes content, padding and border

Note

In general,

- Margins of horizontally aligned elements cumulate
- Margins of vertically aligned elements collapse
- box-sizing: border-box is preferred by the developers

Background

- background: *bg-color bg-image position/bg-size bg-repeat bg-origin bg-clip bg-attachment* initial|inherit;
- background-color: *color*|transparent|initial|inherit;
- background-image: *url*|none|initial|inherit;
- background-position: *value/xpos ypos/ x% y%*;
- background-size: auto|*length*|cover|contain|initial|inherit;
- background-repeat: repeat|repeat-x|repeat-y|no-repeat|initial|inherit;
- background-origin: padding-box|border-box|content-box|initial|inherit;
- background-clip: border-box|padding-box|content-box|initial|inherit;
- background-attachment: scroll|fixed|initial|inherit;

Font

- font: *font-style font-variant font-weight font-size/line-height font-family* | initial | inherit;
- font-style: normal | italic | oblique | initial | inherit;
- font-variant: normal | small-caps | initial | inherit;
- font-weight: normal | bold | bolder | lighter | *number* (100-900) | initial | inherit;
- font-size: medium | xx-small | x-small | small | large | x-large | xx-large | smaller | larger | *length* | initial | inherit;
- line-height: normal | *number* | *length* | initial | inherit;
- font-family: *font* | initial | inherit;

Font family

Serif

- Georgia, Palatino Linotype, Book Antiqua, Times New Roman, Times etc.

Sans-Serif

- Arial, Helvetica, Arial Black, Gadget, Comic Sans MS, cursive, Impact, Charcoal, Lucida Sans Unicode, Lucida Grande etc.

Monospace

- Courier New, Courier, Lucida Console, Monaco etc.

Text

- text-align: left | right | center | justify | initial | inherit;
- text-decoration: none | underline | overline | line-through | initial | inherit;
- text-indent: *length* | initial | inherit;
- text-overflow: clip | ellipsis | *string* | initial | inherit;
- text-shadow: *h-shadow v-shadow blur-radius color* | none | initial | inherit;
- text-transform: none | capitalize | uppercase | lowercase | initial | inherit;
- vertical-align: baseline | *length* | sub | super | top | text-top | middle | bottom | text-bottom | initial | inherit;
- direction: ltr | rtl | initial | inherit;

Table

- border-collapse: separate | collapse | initial | inherit;
- border-spacing: *length* | initial | inherit;
- caption-side: top | bottom | initial | inherit;
- empty-cells: show | hide | initial | inherit;

List

- `list-style: list-style-type list-style-position list-style-image | initial | inherit;`
- `list-style-image: none | url | initial | inherit;`
- `list-style-position: inside | outside | initial | inherit;`
- `list-style-type: value;`
- Ex:
 - `ul.a {list-style-type: circle;}`
 - `ul.b {list-style-type: square;}`
 - `ol.c {list-style-type: upper-roman;}`
 - `ol.d {list-style-type: lower-alpha;}`
 - `ul { list-style-image: url('sqpurple.gif'); }`

Display

Inline: Default value. Displays an element as an inline element (like)

Block: Displays an element as a block element (like <p>)

Inline-block: Display inline element which can accommodate height and width

None: The element will not be displayed at all (has no effect on layout)

Initial: Sets this property to its default value.

Inherit: Inherits this property from its parent element.

```
<html>
<head> <style>
p {
  display: inline;
} </style>
</head>
<body> <p>This is a paragraph.</p> <p>This is a paragraph.</p><p>This is a paragraph.</p>
<p>This is a paragraph.</p> <p>This is a paragraph.</p>
</body>
</html>
```

Positioning

- **Absolute** - Positioned relative to the first parent element that has a position other than static.
- **Fixed** - Positioned relative to the browser window.
- **Relative** – Positioned relative to its normal position
- **Static** – Default - No position, the element occurs in the normal flow (ignores any top, bottom, left, right, or z-index declarations)
- **Inherit** – The value of the position property should be inherited from the parent element

Float

- **Clear:** Specifies on which sides of an element where floating elements are not allowed to float
 - Left, right, both, none, inherit
- **Float:** Specifies whether or not an element should float
 - Left, right, none, inherit

Overflow

- **overflow**: Specifies what happens if content overflows an element's box
- **overflow-x**: Specifies what to do with the left/right edges of the content if it overflows the element's content area
- **overflow-y**: Specifies what to do with the top/bottom edges of the content if it overflows the element's content area
 - visible, hidden, auto, scroll

Media queries

Media queries is a CSS3 module allowing content rendering to adapt to conditions

- 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

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 (*mediafeature* and|or|not *mediafeature*) {
CSS-Code;
}

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

Some media features

- Min-height
- Min-width
- Max-height
- Max-width
- Height
- Width
- Color
- Orientation

Set the view port

- Pages optimized for a variety of devices must include a meta viewport tag in the head of the document. A meta viewport tag gives the browser instructions on how to control the page's dimensions and scaling.
- Use the meta viewport tag to control the width and scaling of the browser's viewport.
- Include width=device-width to match the screen's width in device-independent pixels.
- Include initial-scale=1 to establish a 1:1 relationship between CSS pixels and device-independent pixels.
- **<meta name="viewport" content="width=device-width, initial-scale=1">**
- <https://webdesign.tutsplus.com/articles/quick-tip-dont-forget-the-viewport-meta-tag-webdesign-5972>
- <https://css-tricks.com/snippets/html/responsive-meta-tag/>

HTML layout elements

- Header
- Nav
- Section
- Article
- Aside
- Footer

HTML layout elements cont..

- **Header**
- The <header> element represents a container for introductory content or a set of navigational links.
- A <header> element typically contains:
 - one or more heading elements
 - logo or icon
 - authorship information.
- **Nav**
- The <nav> tag defines a set of navigation links.
- Notice that NOT all links of a document should be inside a <nav> element. The <nav> element is intended only for major block of **navigation links**

HTML layout elements cont..

Section

- The <section> tag defines sections in a document, such as chapters, headers, footers, or any other sections of the document.

Article

- The <article> tag specifies independent, self-contained content.
- An article should make sense on its own and it should be possible to distribute it independently from the rest of the site.

HTML layout elements cont..

Aside

- The <aside> tag defines some content aside from the content it is placed in.
- The aside content should be related to the surrounding content.

Footer

- The <footer> tag defines a footer for a document or section.
- A <footer> element should contain information about its containing element.

Example for CSS pseudo-classes

- Pseudo-classes allow the selection of elements based on state information that is not contained in the document tree.
- The syntax of pseudo-classes:
 - `selector:pseudo-class {property:value;}`
- CSS classes can also be used with pseudo-classes:
 - `selector.class:pseudo-class {property:value;}`
- Example

```
a:link {color:#FF0000;}    /* unvisited link */
a:visited {color:#00FF00;} /* visited link */
a:hover {color:#FF00FF;}  /* mouse over link */
a:active {color:#0000FF;} /* selected link */
```

Some more Pseudo classes

- :checked
- :disabled
- :empty
- :enabled
- :first-child
- :last-child
- :focus
- :valid
- :invalid
- :optional
- :required
- :read-only
- :read-write

Pseudo elements

Pseudo-elements represent entities that are not included in HTML.

Eg:

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

Some pseudo elements

- ::after
- ::before
- ::first-line
- ::first-letter
- ::selection

CSS Combinators

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

There are four different combinators in CSS:

- **Descendant selector** (space): matches all elements that are descendants of a specified element.
- **Child selector** (>): selects all elements that are the immediate children of a specified element.
- **Adjacent sibling selector** (+): selects all elements that are the adjacent siblings of a specified element.
- **General sibling selector** (~): selects all elements that are siblings of a specified element.

CSS Combinators example

```
<style>
.div1 p {
  color: yellow;
}
.div2+p {
  color: red;
}
.div3>p {
  color: green;
}
.div4~p {
  color: blue;
}
</style>
```

```
<div>
<div class="div1">
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <span><p>Paragraph 3 </p></span>
</div>

<p>Paragraph 4</p>
<p>Paragraph 5</p>
</div>
```


Transform

- Allows you to visually manipulate an element by skewing, rotating, translating, or scaling CSS3 supports 2D and 3D transformations.
- Prefixes or notations according to browser support
 - -ms- (Internet Explorer)
 - -webkit- (Chrome, Safari, Opera)
 - -moz- (Firefox)

Methods

- **translate()** : method moves an element from its current position
- **rotate()**: method rotates an element clockwise or counter-clockwise according to a given degree.
- **scale()**: method increases or decreases the size of an element, according to the width and or height.
- **skew()**: method skews an element along the X-axis and or Y-axis by the given angle.

Examples

- `div {
 -ms-transform: translate(50px,100px); /* IE 9 */
 -webkit-transform: translate(50px,100px); /* Safari */
 transform: translate(50px,100px);
}`
- `transform: rotate(20deg);`
- `transform: scale(2,3);`

Transition

- It is CSS effect which changes style of element w.r.to time.

Properties:

- **transition-property**: Specifies the name of the CSS property the transition effect is for
- **transition-duration**: Specifies how many seconds or milliseconds a transition effect takes to complete
- **transition-timing-function** : Specifies the speed curve of the transition effect
 - linear, ease, ease-in, ease-out, ease-in-out and cubic-bezier(n,n,n,n)
- **transition-delay**: Specifies a delay (in seconds) for the transition effect

Example

```
div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  transition-property: width;  
  transition-duration: 2s;  
  transition-timing-function: linear;  
  transition-delay: 1s;  
}  
  
div:hover {  
  width: 300px;  
}
```

Short-hand

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

Animation

Allows animation of most HTML elements without using Scripts

Properties:

- **@keyframes** : defines style change rules for elements
- **animation-name**: Specifies the name of the @keyframes animation
- **Animation-duration**: Specifies how many seconds or milliseconds an animation takes to complete one cycle
- **Animation-timing-function**: Specifies the speed curve of the animation
- **Animation-delay**: Specifies a delay for the start of an animation
- **Animation-iteration-count**: Specifies the number of times an animation should be played
- **Animation-direction**: Specifies whether an animation should play in reverse direction or alternate cycles
 - Normal, reverse, alternate, alternate-reverse
- **Animation-play-state**: Specifies whether the animation is running or paused
 - Running or paused

Example

```
.circle
{
  width:200px;
  height:300px;
  position:relative;
  background:Yellow;
  animation-iteration-count:infinite;
  animation-timing-function:linear;
  animation-name:moovcircle;
  animation-duration:5s;
}
@keyframes mooncircle
{
  0%{left:0px; top:0px;}
  100%{left:500px; top:500px; border-radius:50%; }
}
<div class="circle"></div>
```

Flex Box

- The Flexible Box Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning
- **Display:Flex**
 - Displays an element as a block-level flex container

Flex Box..

- **Flex-grow:** specifies how much the item will grow relative to the rest of the flexible items inside the same container
 - *number*|initial|inherit;
- **Flex-shrink:** specifies how the item will shrink relative to the rest of the flexible items inside the same container
 - *number*|initial|inherit
- **Flex-basis:** specifies the initial length of a flexible item
 - *number*|auto|initial|inherit
- **Flex:** sets the flexible length on flexible items
 - *flex-grow flex-shrink flex-basis*|auto|initial|inherit;

Flex Box container..

- **Flex-direction**

- defines in which direction the container wants to stack the flex items
 - row | row-reverse | column | column-reverse | initial | inherit

- **Flex-wrap**

- specifies whether the flexible items should wrap or not
- flex-wrap: nowrap | wrap | wrap-reverse | initial | inherit

- **Flex-flow**

- Is a short-hand for flex-direction and flex-wrap

Flex Box container..

- **Justify-content**

- aligns the flexible container's items when the items do not use all available space on the main-axis (horizontally)
- flex-start | flex-end | center | space-between | space-around | initial | inherit;

- **Align-items**

- specifies the default alignment for items inside the flexible container
- stretch | center | flex-start | flex-end | baseline | initial | inherit

- **Align-self**

- specifies the alignment for the selected item inside the flexible container
- auto | stretch | center | flex-start | flex-end | baseline | initial | inherit
- property overrides the flexible container's align-items property

Flex Box container..

- **Align-content**

- property modifies the behavior of the flex-wrap property.
- similar to align-items, but instead of aligning flex items, it aligns flex lines
 - stretch | center | flex-start | flex-end | space-between | space-around | initial | inherit;

Transform attributes

- Transform-origin
 - property allows you to change the position of transformed elements
 - *x-axis y-axis z-axis* | initial | inherit;
 - **X-axis**- left | center | right | length | %
 - **Y-axis**- top | center | bottom | *length* | %
 - **Z-axis**- length
- Transform-style
 - property specifies how nested elements are rendered in 3D space
 - flat | preserve-3d | initial | inherit;