**C.S.S**

**What is C.S.S?**

CSS stands for Cascading Style Sheet. It is called cascaded because it follows the water form model, that is, it follows a step by step approach from top to bottom and the most recent rule/declaration gets accept and the former canceled/ignored.

CSS contains series of rule sets which is applied to your HTML document to beautify it or make it look fancy. Such giving is background, text color, font, different font size etc.

**Linking CSS to your HTML**

**1. External CSS:**

It is called external CSS because a separate CSS file/sheet is created and linked to your HTML document.

**Steps**

Create your CSS file and save it with the .css extention

Goto our HTML document and link the css sheet using the link tag

<link rel="stylesheet" href="css-file-directory/css-filename.css" />

**2. Internal CSS:**

It is called internal CSS because the CSS rules are written inside your HTML document. Using the <style></style> tag

**3. Inline CSS:**

This type of CSS is written in the opening tag of the element you want to style. Using the style attribute.

**Example:**

<p style="font-size: 20px;">content</p>

**CSS Rule Set Format**

**Selector {**

**Property(s): value(s);**

**}**

**Selector:** is the element you want to style.

**Selector Types**

1. Element selector: e.g Body, p, h1 etc

2. Class selector: e.g .heading

3 Id selector: e.g #title

4. Group selector: e.g h1, h2,

5. Nested which is further divided into two; direct child and descendant selector.

i.) Direct child selector (using the > sign): locating an element nested inside of another e.g. (span p) who is a direct child not just a descendant. In this case p is a direct child of span.

ii.) Descendant selector: locating an element nested inside of another e.g. (div span p) In this case span and p are descendants of div.

6. Universal selector > \* this selects everything on the page, it is basically used for resetting the browser's default page settings.

**Order of Precedence**

**Specificity** = order precedence/importance

!important > ID selector > class selector>element selector>universal

**Comment in CSS**

Comments are personal information which are ignored by the browser, it is to aid the understanding of the CSS rule or to clarify other users. It is like a self-note

**Syntax:**

/\* comment goes here \*/

**Colors**

* **Color name**

There are about 140 color names in html. And VS code helps by providing suggestions of the color names in alphabetical order.

* **RGB** (red green blue) (255, 255, 255) (50%, 50%, 50%)

Rgba (red green blue alpha/transparency) (255, 255, 255, 1.0)

* **HEX** #000 #000000

**Choosing Colors**

>coolors.co (tools > contrast checker)

>webaim.org

Here you enter your forecolor and the background to get your ratings and how compatible they are.

**CSS Units**

1. **Absolute units:** are fixed and (mostly) relate to some physical measurement. Once they are declared, their size cannot be altered by changing the font size of some other element.

**Examples:**

* Centimeters (cm) - 1cm = 37.8px = 25.2/64inches
* Millimeters (mm) - 1mm = 1/10 of 1cm
* Inches (in) - 1in = 2.54cm = 96px
* Picas (pc) - 1pc = 1/6 of 1in, 1pc – 12pt
* Points (pt) - 1pt = 1/72 of 1in
* Pixels (px) - 1px = your viewport pixel, a single dot = 1/96 of 1in

**Note:** Most of these absolute units are useful when used for print rather than screen output. For example, we don’t typically use centimeter (cm) on screen. The only absolute unit that we commonly use is the pixels (px).

1. **Relative units:** do not have an objective measurement. Instead, their actual size is determined by the size of a parent element. This means that their size can be altered by changing the size of the dependent element.

**Examples:**

* Em – em is relative to the size of the parent element in the case of typographical properties like font-size, and other properties like width. E.g. if the browser’s font size is 16px then 1em = 16px, 2em = 32px.
* Rem (root em) - rem is relative to the html (root) font size.
* Percent (%) - is relative to the root (default; html) element size.
* Vw (viewport’s width) - 1% or 1/100 of the viewport’s width (screen).
* Vh (viewport’s height) - 1% or 1/100 of the viewport’s height (screen).

**Link:**

1. developer.mozilla.org/enUS/docs/learn/CSS/building\_blocks/values\_and\_units

2. bitsofco.de/css-font-sizing.

**Box Model**

4 values = |top|, |right|, |bottom|, |left|.

3 values = |top|, |right & left|, |bottom|.

2 values = |top & bottom|, |right & left|.

1 values = |top right bottom left|.

**Border** {top, right, bottom, left; style: ridge, groove, dotted, dashed, solid, and double; width; color}

**Outline** {style: ridge, groove, dotted, dashed, solid, double, width, color, offset(+ve & -ve)}

**Padding:** inner spacing of the content **{**size**;** 5px**}**

**Margin:** outer spacing of the container **{**size**;** 5px**}**

**Margin-**left & right: auto for centering block elements (text-align for inline)

**Border-radius:** curved edge

**Box-shadow:** {|x-axis-value| |y-axis-value| |blur-radius| |spread-value| |color|}

**Box-sizing:** {border-box, content-box}

**Border-box** - subtracts the paddings/border from the width/height of an item and assigns the remaining size to the content of that item.

**Content-box** - is the default value, the padding/border are added to the content of that item which then increases its width and height.

**CSS Typography**

Typography is the way that text is arranged and presented.

Text-decoration {underline, overline, none, line-through}

Text-align {left, right, justify, center}

Text-transform {lowercase, uppercase, capitalize}

Text-indent {size; 10px, 2em, 3rem}

Text-shadow: {|x-axis-value| |y-axis-value| |blur-radius| |color|}

Line-height {default value is 1.2}

Letter-spacing {size; 2px, 0.1em, 0.2rem}

word-spacing { size; 3px, 0.5em, 0.6rem }

font-size { default value is 16px, 20px, 1em, 2rem }

font-weight { 100 to 900, lighter, light, normal, bold, bolder }

font-style {italic, oblique, normal}

font-variant {normal, small-caps, initial, inherit}

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.

font-family {default is serif, sans-serif, monospace, cursive, fantasy etc are generic family}

**Imported fonts:** {linked before the link tag in html document or the first item on your CSS sheet}

Shorthand for font

Font {|font-style| |font-variant| |font-weight| |font-size/line-height| |font-family|}

**PSEUDO CLASSES VS PSEUDO ELEMENTS**

**Pseudo-class/element**A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected element(s)

Pseudo classes basically deal on the state of element whereas pseudo elements act as if extra element is being added to that element which is selected.

Also pseudo classes starts with a colon (:) while pseudo classes starts with double colons (::)

: for pseudo-class

:: for pseudo-element

Pseudo Classes

**STYLING CSS LINKS USING PSEUDO- CLASS**

A:link {styling the actual link(anchor) the way you may want it}

a:visited {styling the link(anchor) the way you may want it after it has been visited}

a:hover {styling the link(anchor) the way you may want it when it is hovered}

a:focus {usually paired with hover, this is applied when the user is using tab key}

a:active {styling the link(anchor) the way you may want it once it is being clicked}

**OTHERS:**

:not()

:first-child

:last-child

:only-child

:nth-child(odd, even, numbers,)

Pseudo Elements

::after pseudo element: is used to place an extra content after the specified element. And the content does not necessarily have to be text and is not treated as text.

::before pseudo element: is used to place an extra content before the specified element.

::first-letter: selects the first letter of the specified element.

**STYLING LISTS**

List-style-type{decimal, decimal-leading-zero, lower-alpha, upper-roman, none etc}

List-style-image{url for your image}

List-style-position{outside, inside}

Using some attributes such as type, start, value, reversed.

Using ::marker pseudo-element{color, font-family, font-size, content(as string) etc}

Shorthand for styling list

List-style {type image position}

**Link**

<https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes>

**CSS DISPLAY TYPES**

**Block:** has a 100% width assigned to them (parent), they stack on top of each other.

With block element, you can apply height, margin, padding, border, and outline.

**Inline:** this doesn’t allow the application of height, margin. Padding on inline doesn’t work similarly to the block display.

**Inline-block:** takes the property of a block element while retaining that of the inline display. You can keep the contents inline, and also add properties such as height, margin, padding, border, etc.

**None:** this makes the item not to shown; though this is different from visibility/opacity property as the display property takes the element off the page completely and can’t be read by a screen reader.

Others are; grid and flexbox, which will be discussed later.

**Similar PropertieS**

**Visibility:** {hidden, visible}. Set if an element will be seen or not. Nevertheless, it can still be read by a screen reader.

**Opacity:** {0 to 1}. Set how transparent an element can be.

**CSS POSITION**

CSS position property sets how an element is positioned in a document.

**Static:** The element is positioned according to the normal flow of the document. The [top](https://developer.mozilla.org/en-US/docs/Web/CSS/top), [right](https://developer.mozilla.org/en-US/docs/Web/CSS/right), [bottom](https://developer.mozilla.org/en-US/docs/Web/CSS/bottom), [left](https://developer.mozilla.org/en-US/docs/Web/CSS/left), and [z-index](https://developer.mozilla.org/en-US/docs/Web/CSS/z-index) properties have no effect. This is the default value.

**Relative:** this type of positioning is in consideration of the parent container (nested) without having to position the parent container relative.

**Absolute:** positions the item relatively to the parent element. If there is no parent that is set to position relative then it takes the initial boundary which is the browser’s window.

**Fixed:** this content of the element becomes fixed at the specified position assigned to it. Even when the page is scrolled, the content remains fixed.

**Sticky:** it is similar to fixed because it will stay on its normal flow until it reaches the spot that you have defined. And scrolls away when the parent container reaches its height and then it scrolls away.

**CSS LAYOUTS**

**Using Grid:**

The grid display just as the flexbox is made element is made up of two main parts: the container (parent, where the display is set to grid) and the items (child, which make up the content of the box). Here, when the display is set to “grid”, the items appear as a block; one on top of another unlike the flexbox.

On The Container

**Display**: {grid}

**Grid-auto-flow**: just like the flex-direction, is used to set how the item will be arranged. {Row, Column}

**Grid-template-columns**: this property is used to set the width of the columns. It can be specified using size-values such as 300px, 1fr [repeat (6,1fr)], and auto. The number of times it is specified is the number of columns it will create.

**Grid-template-rows:** this is used to set the number of rows for the container.

**Row-gap:** gives a horizontal space for the rows.

**Column-gap:** gives a vertical space for the columns.

**Gap**: {size; this gives extra spaces between the grid items horizontally and vertically}

Gap shorthand

Gap: |row-gap| |column-gap|; or gap: |row-gap & column-gap |;

**Justify-content:** aligns the content horizontally on the container {start, end, center, space-around, space-between, space-evenly}

**Align-items:** aligns items vertically on the container {start, end, center, stretch, baseline (same baseline of its content (text) no matter the individual height)}

**Align-content**: just like justify content, it aligns the rows, visible if there are more than one row. {start, end, center, space around, space-between, space-evenly}

On The Item

**Grid-column-start:** {number}. Species the vertical starting point of the selected item

**Grid-column-end:** {number}. Species the vertical ending point of the selected item.

**Grid-row-start:** {number}. Species the horizontal starting point of the selected item.

**Grid-row-end:** {number}. Species the horizontal ending point of the selected item.

Shorthand for grid-column/row-start/end

**Grid-column:** start-value / end-value;

**Grid-row:** start-value / end-value;

A**lternatively**, you can use span to join number of gridlines column or rows together.

Grid-column: 1 / span 4; or

Grid-row: span 3;

**Using Flexbox**

The flexbox element is made up of two main parts: the container (parent, where the display is set to flex) and the items (child, it makes up the content of the box). Here, when the display is set to “flex”, the items appear as a inline-block; side by side unlike the grid.

On The Container

**Display**: {flex}

**Gap**: {size; this gives extra spaces between the flex items}

**Flex-direction** {row/column [column-reverse/row-reverse][the arrangement of the items on the container]}x-

**Flex-wrap** {nowrap, wrap, wrap-reverse [makes item to go to new line when it reaches its width]}

Flex Flow Shorthand

Flex-flow: |flex-direction| |flex-wrap|;

**Justify-content:** aligns the content horizontally on the container {flex-start, flex-end, center, space-around, space-between, space-evenly}

**Align-items:** aligns items vertically on the container {flex-start, flex-end, center, stretch, baseline (same baseline no matter the individual height)}

On The Item

**Order:** specifies the order of arrangement of individual items {number}. By default all the item are of order 0, when order is set to a positive (+ve) value, without setting for other items it becomes the last, unless when the order is set to negative (–ve) value.

**Flex-basis**: defines the minimum width of a flex item {size}

**Flex-grow**: defines the proportion upon which items can grow or increase {number: default is zero} has to work with flex wrap

**Flex-shrink**: defines the rate at which items can shrink {numbers: default value is 1, zero to remove shrink. The flex-wrap has to be nowrap}

Flex shorthand

Flex: grow shrink basis (two value will be for grow and flex-basis);

Styling Images

As background: Where image can be used as wallpaper upon which other items can be placed, you can add the image by selecting the item you want to add the background to, and then use the following properties:

Background-color: {color} as a fallback option in case the image is unable to load.

Background-image: {(url(‘folder/image-name.image-format’);} to actually add the image.

Background-repeat: {repeat, repeat-x, repeat-y, space, round, no-repeat} to avoid the image from repeating as the background.

Background-position: {|x-position| |y-position|} or {x & y position} {center, right, bottom, left, top} to specify where the image background should be positioned.

Background-size: {contain, cover, value} to specify the size; i.e. how the image should appear on the background.

Background-attachment: {fixed, scroll}is used to control if a background image is scrollable or not.

**Background Shorthand**

Background: |color| |image| |repeat| |position/size|;

**GRADIENT**

This has to deal with the combination of two or more colors.

Linear-gradient:

**Direction**: direction you want the color to start from. [(|to left|, |to right|, |to top|, |to bottom|, |to top right|, |to top left|, |to bottom right|, |to bottom left,|); optional]. You can also specify the directions in degrees (0deg is top), (90deg is right), (180deg is bottom)

**Color-stops**: these are the different colors you want to add to the gradients (steelblue, gray, white). You can also specify the percentage of the colors. E.g: (green 29%, red 50%, 31%).

**Repeating-linear-gradient**: will make the gradient appear multiple times while specifying the different percent for the colors. E.g:

Repeating-linear-gradient(red, yellow10%, green 20%);

Radial-gradient:

**Shape:** defines the shape of the gradient. {ellipse (default), circle}

**Position**: defines the position of the gradient. {at center(default),at left, at right, at bottom, at top}.

**Color-stops**: these are the different colors you want to add to the gradients (steelblue, gray, white). You can also specify the percentage of the colors. E.g: (green 29%, red 50%, 31%).

**Example**:

**Background**: radial-gradient (circle at top, blue, grey 30%, yellow 15%, green 60%);

Conic-gradient:

Conic gradient is a type of gradient rotated around a center point.

Conic-gradient([from angle] [at position,] color[degree|percent], color[degree|percent], …); E.g:

Background: conic-gradient(from 60deg at center, red 60deg, yellow 45deg, green);

**Note:** You can combine image with colors by using linear-gradient. Example:

Background: linear-gradient(|to direction[(|to left|, |to right|, |to top|, |to bottom|); optional]|steelblue, gray, white), url(“image-directory/image-name.image-format”);

To make a text have an image filled as color:

Firstly, make the image as background for the text.

Then use background-clip property with a value of “text”, while using different browsers’ prefix to have a wide range support from different browsers.

Next, give it color with transparency so as to have a clear view of the image.

**MEDIA QUERIES**

These are set of rules that help to create responsiveness on different screen sizes of different devices.

Syntax:

@media |media type| and (condition(s): breakpoint) {

CSS Rules

}

Min-width = starting from (and above).

Max-width = up to

Examples:

@media screen and (min-width: 481px) {

Background: red;

}

@media screen and (max-width: 681px) {

Background: green;

}

Min-width: the styles should apply from the specified size upwards/above

Max-width: the styles should apply from the specified size downwards/below

@media screen and (orientation: landscape) {

Background: cyan;

}

@media screen and (min-aspect-ratio: 7/4) {

Background: grey;

}

@media screen and (max-height: 425px) and (min-aspect-ratio: 7/4) {

Background: magenta;

}

**Variables**

Variable are used as value holders or containers that we can call upon when we need such value. In CSS we use variables (mostly for colors) to avoid repetition and for ease access and correction when needed instead of having to correct the value in all instance we have used such value.

We declare the variable using the keyword (:root {--variable-name: variable-definition(or value)}). And call the variable using “var(variable-name)”

Example

:root{

--Light-Red-Bgcolor: #00f;

--Dark-Red-Bgcolor: #006;

--Large-Font-Size: 2rem;

}

P{

background-color: var(--Dark-Red-Bgcolor);

font-size: var(--Large-Font-Size);

}

**TRANSFORMATION**

This property is used to apply different transformation style to the selected element. Transformation styles may include;

1. Translate: is movement from one point to another.

translateX(50%); - this tries to move the element 50% of its width. +ve value moves it to the right while -ve value moves it to the left.

translateY(4rem); - this tries to move the element up of the specified value. +ve value moves it down while -ve value moves it up.

Shorthand for the translate

Translate(x-value, y-value);

1. Rotate: is movement around an axis.

rotateX(angle-in-degree); - this tries to move the element the on the angle that you have specified on the x-axis(up-down). 45deg is slant, 90deg is eye-level, 180deg is upside down.

rotateY(angle-in-degree); - this tries to move the element the on the angle that you have specified on the y-axis(left-right). 45deg is slant, 90deg is eye-level, 180deg is opposite direction.

rotateZ(angle-in-degree); - this tries to move the element the on the angle that you have specified on the z-axis(clockwise). 45deg is rolled aslant, 90deg is top-level, 180deg is opposite direction. The rotate(angle-in-deg) is similar to rotate(angle-in-deg).

1. Scale: is used to either increase or decrease the size of an element by the specified value.

scaleX(value); - increase the or reduces the size of the element on the left and right sides only.

scaleY(value); - increase the or reduces the size of the element on the top and bottom sides only.

Shorthand for scale

scale(x-value, y-value); or scale(x&y-value);

1. Skew: is used to slant an element on it sides. Or to make an object go sidling.

SkewX(angle-in-deg): is used to slant an element from the left and right.

SkewY(angle-in-deg): is used to slant an element from the top and bottom.

Shorthand for skew

skew(x-value, y-value);

**TRANSITION**

Has to do with different changes we can apply to an element at a particular time frame.

1. Transition-delay: the time period (usually in seconds) that you want the transition to wait before it can commence.
2. Transition-duration: the time period you want your transition to last.
3. Transition-property: the particular property of the element that you want the change to apply.(it can be a specific property or it can be all).
4. Transition-timing-function: this is the style of transition that you want to apply. (linear, ease(default), ease-in, ease-out, ease-in-out).
5. Linear: same speed from start to end.
6. Ease (default): slow start, then fast and then end slowly.
7. Ease-in: slow start.
8. Ease-out: slow end.
9. Ease-in-out: slow start and end slow.

Shorthand for transition

Transition: |property| |duration| |timing-function| |delay|;

Transition: property 2s, property 3s;

**ANIMATION**

Has to do with different changes (in appearance, motion, e.t.c) we can apply to an element at a particular time frame.

1. Animation-name: this is the user-defined name for the animation you intend to create.
2. Animation-delay: the time period (usually in seconds) that you want the animation to wait before it can commence.
3. Animation-duration: the time period you want your animation to last.
4. Animation-timing-function: this is the style of animation that you want to apply. (Linear, ease (default), ease-in, ease-out, ease-in-out).
5. Animation-iteration-count: this is the number of times you want the animation to repeat (number, infinite).
6. Animation-direction: is the direction you want the animation to follow (normal, alternate, reverse, alternate-reverse).

Animation-direction can have the following values -

(1) Normal: forward/normal direction, this is the default value.

(2) Reverse: the animation sets in the reverse direction (backward/opposite).

(3) Alternate: the animation plays normal first and then reverses.

(4) alternate-reverse: the animation plays reverse first and then normal.

Example:

            div {

                width: 100px;

                height: 150px;

                background-color: green;

                animation-name: text;

                animation-duration: 2s;

                animation-iteration-count: 4;

                animation-direction: alternate;

            }

            @keyframes text {

                from { margin-left: 60%;}

                to { margin-left: 0%; }

            }

**Keyframes:**

The animation keyframe is used to specify the changes that will occur at different time frames (period) usually in percent (%) or by using these keywords “from” and “to”.

**Syntax**

@keyframes identifier or animation-name {

Period1{

Property: value;

}

Periodn{

Property: value;

}

}

1. Animation-fill-mode: {forwards, backwards(normal)}this usually applies after the end of the animation, where the property (background-color, font-size) is been set to either the last user specified property (background-color, font-size) [in this case the last time period which is the 100% (must be 100% or “to”)] or to the default.

NOTE: The animation iteration-count that are odd counts are odd won’t work with animation-fill-mode.

Example:

@keyframes rotation{

      50%{

          font-size: 30px;

          background-color: green;

      }

      100%{

          font-size: 50px;

          background-color: pink;

      }

   }

Shorthand for animation

Animation: |duration| |timing-function| |delay| |iteration-count| |direction| |fill-mode| |identifier or animation-name|;

**Organization**

Create and follow a pattern you will like to organize your CSS code.

You may want to style based on the order of appearance of element in the html document. So you can easily trace the code.

You may decide to style some element in groups such as styling all the general and default settings. Or any way that suits you. Or if your team has an approach already then follow it.

Always use comments as a heading for some styling that may be as a group or section for easy access.

Make sure to always use same letter case (i.e. if you are used to uppercase then be consistent with it or otherwise).

Use variables for values that will appear multiple times when possible for easy modification or changes to your code.

Try as much as possible to apply the DRY (Don’t Repeat Yourself) principle. i.e. don’t repeat selectors, variables, etc. Try to group selectors where possible.

Neatly format your code.

Sort your properties alphabetically. This should be when you are through with a particular section. Highlight the properties and sort. This will help you to know if you repeated a property twice or more and probably those properties are not getting you what you want because cascading have been applied to the properties. (Especially, when dealing with big projects). If you don’t like this method you can try arranging the properties orderly as you want it to effect, so you can easily trace the property for changes or modification.