CSS Cheat Sheet

**Basic Styling**

Define properties of stuff like p, h1, etc

List of things you can manipulate

font-family, font-size, font-weight, color, background-color, opacity, text-transform (*lower and uppercase*), text-align, position, line-height, padding, margin, background-image, list-style, border-radius,

p {

font-family: Arial;

color: maroon;

}

Can also create custom class names using .

.paragraph {

padding-left: 20px;

}

Then on Html file reference like

<div class= “paragraph”>

</div>

You can technically have multiple classes in a single element

<p class= “paragraph greenbold”>Green Bold Text</p>

You can also create custom class version of existing attributes

h1.paragraph {

font-size: 50%;

}

<h1 style= “paragraph”>Big Font</h1>

Nested elements can be written like

.paragraph h1 {

font-size: 50%

}

<div class= “paragraph”>

<h1>Big Font</h1>

</div>

You can create unique styles for those with an id using #

#large-title {

font-size: 50%;

}

If multiple elements have the same style, just use the comma

h1, h2.paragraph, .paragraph h3, p, .paragraph, #bruh {

font-family: Arial;

}

**UNITS**

Existing units when defining value are px, %, em, rem.

px is like a fixed value that isn’t responsive, therefore it is called a static unit.

The other three are responsive, therefore are defined as dynamic units.

em represents the font-size of the current element or the default base font-size set by the browser.

If the default is 16px then 1 em = 16px;

rem stands for root em where it checks the root element for the font-size.

% is basically size relative to the parent element.

**COLOR**

Basically Hexcode is written as #vvv or #vvvvvv colors for red, green, then blue

Can be written as rgb(v, v, v); with values from 0 – 255;

With color, you can also adjust the hsl aka hue, saturation, and lightness

color: hsl (120, 60%, 70%);

For hue values, 0/360 is red, 60 is yellow, 120 is green, 180 is aqua, 240 is blue, and

purple is 300. Adjust accordingly to the degree of the color wheel.

Saturation refers to the intensity of the color.

Lightness refers to the lightness of the color.

rgba() refers to rgb with the last value called alpha, or opacity

Between 0-1, we can determine how much of the of the element we want to see, where 0 is transparent. You can also write color: transparent; to do the same thing as rgba(v, v, v, 0);

**FONT**

font-family defines the font family used.

You can also define another font-family as a backup plan followed by ,serif or ,sans-serif.

font-weight defines the font weight used

the values can be words or from 100-900.

400 is usually default, 300 is usually light, and 700 is usually bold.

font-style refers to the font style used

There is italic, strong, etc.

word-spacing refers to the spacing between words.

You would use values, such as em, to define the amount of space in-between words.

letter-spacing refers to spacing between letters.

text-transform refers to the styling of the text;

it can use uppercase, lowercase, etc.

text-align refers to the position of the text relative to the space it is within.

It can be defined with left, center, right, etc.

line-height refers to the height of the line, which is the height of the font + height in-between fonts vertically.

Serif vs Sans serif.

Serif are defined as fonts with fancy details to make them stand out more compared

to other fonts.

Sans means to exclude, so excluding serif would be defined as fonts that remove anything fancy

is seen as minimalist and clean. These are ideal when using small text.

In HTML you can link fonts using the <link> followed by the url code.

In CSS you need to use the @font-face property. You can get it by copying and pasting the url into the web browser and look for the ones labeled /\* or latin \*/.

**STRUCTURE**

padding and margin are the two main props used to manipulate the element’s space;

they can be written in 4 values, top, right, bottom left.

For example, a padding with 50 px top and bottom with 25px left and right can be written as

padding: 50px 25px 50px 25px; or padding: 50px 25px; If all 50px then padding: 50px;

padding refers to in-between-space within the element.

margin refers to in-between-space outside of the element.

You can add auto to adjust the margin

margin: 0 auto means top and bottom 0px and adjust left and right to center;

It will not auto if the width of the element is not set.

borders refer to the border of the element

You can list three attributes for border. thickness, style, color;

border: 3px solid black;

border-radius basically gives an element a circular look

border radius can be defined in px or %

100% = circle

box-shadow gives an object a “shadow: by taking its border color and shifting it to the desired axis.

box-shadow: #px #px uses two values for the x and y offset values.

box-shadow: 0 4px;

min-width/max-width/height defines the minimum and max size of an element to adjust for responsive

web design.

overflow refers to a property to show/not show the entire image in case the element size is not large

enough.

You can either use hidden, scroll, or visible.

visibility refers to show or hide the element. Uses visible and hidden.

**Positioning**

position defines how to arrange elements in a certain manner.

static is default, relative is basically the positioning relative to it’s original static position,

absolute is basically you defining the element’s location regardless of the other elements.

fixed is similar to absolute, except fixed keeps its position regardless of how much you

scroll.

z-index refers to which elements are on top of each other incase they intersect. Think of layers from PS;

z-index

display refers to how much space we want to use to showcase our elements.

inline would try to use the least amount of space to display the element.

block would fill the entire width of the webpage by default.

You cannot put two blocks next two each other.

inline-block are blocks that can be on the same line. You can also define their width and height.

Note different html tags associate with these type of displays

inline: a, br, button, canvas, em, img, input, label, map, span, strong, video

block: div, figure, figcaption, footer, h1-6, form, header, hr, li, main, nav, ol, p, section, table, ul.

float refers to positioning the element on the webpage.

float left means the element will start its positioning at the left most side, same for float right.

clear is used when floating with multiple elements get messy. It defines how elements behave when

they bump into each other.

clear: left means the element cannot touch any other elements on the left side,

right is same on right side, then there is both and none.

**Flexbox**

display: flex allows the containers to change size and location in response to size of parent container.

Note that flex itself is a block element.

inlin-flex is basically flex but it is now an inline element.

justify-items: helps position flex items

flex-start: Objects positioned from left to right

flex-end: Objects positioned from right to left

center: Objects positioned base on center

baseline: Aligns the baseline

justify-content: helps align content on the x axis

flex-start

flex-end

center

space-around: objects will be positioned with equal space with space before and after element.

space-between: objects will be positioned with equal space without the extra space.

align-items: helps position flex items on the y axis

flex-start Objects positioned from top to bottom

flex-end Objects positioned from bottom to top

center Objects positioned base on the center

baseline: the bottom of the objects will be aligned with each other

stretch: the objects will try to stretch from top to bottom of the container

align-content helps align content on the y axis

flex-start all rows of objects will be positioned at the top of parent

flex-end all rows of objects will be positioned from the bottom

center all rows of objects will be positioned from the center

space-between all rows of objects will be spaced evenly from the top to the bottom of parent.

space-around all rows of objects will be spaced evenly from each other top and bottom.

stretch all rows will be stretched to try to fill parent container

justify-self and align-self is for individual elements that overrides justify-items and align-items

flex-grow: helps define the rate of growth of flex objects.

flex-grow: 2 > flex-grow: 1

flex-shrink: helps define the rate of shrink of flex objects

flex-basis: defines the width of an object before it stretches/shrinks

flex: defines the above 3 in one line.

flex: 0 1 150px = flex-grow: 0; flex-shrink:1; flex-basis: 150px;

flex-wrap helps move flex items to the next line instead of manipulating the actual size to fit in container

wrap helps move child objects/elements down a line if it can’t fit in container

wrap-reverse reverses the order of the wrap display.

nowrap prevents items from wrapping. Default value

flex-direction defines if the flex objects are going to be in row or column to deal with different axis

row: display objects in rows

column: display objects in column

row-reverse: display objects in rows from right to left

column-reverse: display objects in rows from bottom to top

flex-flow declares both flex-wrap and flex-direction

flex-flow: column wrap; Defines there will be wrapping on objects in a column fashion

**INTERACTIONS**

**Psuedo Classes**

**Link Styles**

There are four ways a link can be styled; they are link, visited, hover, and active.

a:link is basically what is being displayed of an unclicked link.

a:visited is what is being displayed of a link after being used.

a:hover is what is being displayed when you hover over a link.

a:active is what is being displayed when you are clicking on a link.

**Others**

Inputs

input:checked selects every checked input element.

input:disabled selects every disabled input element;

input:enabled selects every enabled input element;

[input:focus](https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_focus) selects every input element that has a focus;

input:in-range selects every input element with a value of a specified range.

input:invald selects every input element with an invalid value.

input:optional selects every input element with no “required” attribute;

input:out-of-range selects every input element with a value out of specified range.

input:read-only selects every input element with “readonly” attribute.

input:read-write: selects every input element without “readonly” attribute.

input:required selects every input element with “required” attribute.

input:valid selects every input element with a valid value.

Family

p:empty: selects every p element with no child

p:first-child: selects every p element that is the first child of parent element.

p:last-child: selects every p element that is the last child of parent element.

p:nth-child(n) selects every p element that is the nth child of parent element.

p:nth-last-child(n) selects every p element that is the nth child of parent starting backwards.

p:only-child selects every p element that is the only child of parent element.

Types

p:first-of-type selects every p element that is first element of its kind of its parent.

p:last-of-type selects every p element that is the last element of its kind of its parent.

p:nth-last-of-type selects every nth p element of its kind of its parent starting backwards.

p:nth-of-type selects every nth p element of its kind of its parent.

p:only-of-type selects every p element that is the only p element of its parent.

Difference between child and of-type.

When asking myself this, I found out that you don’t need to specify the element of the child

where you can write the following format .thing :first-child { code } to define the first child.

You can say there is no purpose when it comes to using of-type unless you are using them in

coexistence with the child pseudo classes.

Others

p:lang(word) selects every p element with a lang attribute value starting with word.

:not(p) selects every element that is not a p element.

:root selects the document’s root element.

#thing:target selects the current active #thing element. (clicked on a function to trigger it).

**Pseudo Elements**

p::after inserts something after the content of every p.

p::before inserts something before the content of every p.

p::first-letter selects the first letter of every p.

p::first-line selects the first line of every p.

::selection selects the portion of the element that is selected from the user.

**Combinators**

When referencing multiple elements, we use combinators to detail those with similar properties.

Descendant Selector refers to labeling all elements that are descendants of a specified element.

div p { code; } will create the properties of divs and paragraphs in a div.

Child Selector is similar, but only refers to elements that are a child of the specified element.

div > p { code; } will create the properties of divs and paragraphs that are

direct children to div.

Adjacent Sibling Selector will only refer to elements that are directly near the specified element.

div + p { code; } will create the properties for only the divs and the first paragraph

after div.

General Sibling Selector will refer to elements that are siblings with the specified element.

div ~ p { code; } will create the properties for only divs and the paragraphs that are on the “same level” as div.

**Creating Responsive websites**

**Sizes**

Use rem and em as unit of measure as it would adjust depending on the browser size.

When using images or any other media, use max-width: 100%; and make the height: auto; so that way it will display 100% of the image while adjusting to the size of the page.

When using an image as a background, make sure to consider the following: background-size: cover; background-repeat: no-repeat; background-position: center;

**Media Queries**

Media queries are used to define a website’s layout depending on the size of the screen.

@media only screen and (max-width: 480px) { code } basically dictates the structure.

@media defines when you are doing a media query. only screen refers to the media type being only screen.

max-width helps define when the code will be used. Screens under 480px will be able to view it.

You can also use dpi when you want to display higher quality photos at times.

@media only screen and (min-resolution: 300dpi) { code }

There are also ways to create an OR function within the code. Take the example of the code below

@media only screen (min-width: 320px), (orientation: landscape) { code } you need to put a comma;

Therefore, either the orientation is landscape or min-width is 320 can run this.

**Grid outer elements**

Grids can be used to layout webpages as it is good for aligning and moving elements.

grid is another thing we can use in display where we need to have a grid container as parent elements and grid items as the children.

display: grid displays a grid in a block level while there is display: inline-grid for inline version.

~~grid-template-columns is used to define the columns in a grid. It can take multiple values.~~

~~grid-template-columns: 100px 200px; Defines that there are two columns each fixed with the~~

~~number written. Column1 is 100px and Column2 is 200px~~

~~grid-template-rows is basically row version.~~

grid-template basically is the two mentioned things but can be written in one line.

grid-template: 20px 30px / 20px 10px 30px; values before slash are rows and

values after the slash are the columns.

Unit for grid-template

There is a special unit they use for grid-template and that is fr aka fraction

This is used to prevent the grid from having overflowing issues as it is base off

of the remaining portions of the grid, rather than being fixed values.

grid-template: 2 fr 1fr 1 fr; states that 1st row is 2/4 while others are 1/4

repeat() for grid-template

Repeat basically duplicates the specifications for rows in a given amount of times

repeat(3, 1fr, 2fr) creates 6 rows/columns 1, 3, 5 are 1fr and 2, 4, 6 are 2fr

min-max() for grid-template

Min-max helps define how big a row/column should be in case of browser resizing.

min-max(100px, 500px) defines that the row/column can be min 100px and

max 500px;

grid-gap creates gap inbetween the respective parts of the grid with a value

grid-gap: 20px 10px; basically creates a gap for rows 20px and columns 10px

**Grid-inner elements**

grid-row allows row items to hold multiple rows

grid-row: 1 / 3; basically allows the grid item to take up row 1 and 2, not 3

grid-column is basically same thing as above, but for columns

Note: you can use span to define how many specific rows/columns

grid-column: 1 / span 6; defines you start at column one and occupy 6 columns;

grid-area is basically the grid-row with grid-column

grid-area: 2 / 2 / span 3 / span 3 is basically start row at 2 and start column at 2

and then take up 3 rows and 3 columns.

grid-template-areas helps out when it comes to naming the specific sections

If a grid has 4 columns for example. grid-template-areas: “First First” “Second Second”;

will cause the grid to have two columns named First and two columns named Second.

grid-auto-rows/columns basically gives values of implicitly added rows/columns

grid-auto-rows: 50px; basically give all the extra rows a height of 50 px

grid-auto-flow is basically used to automatically fill either rows or columns.

You can also use the value dense so it will check for rows or columns needed to fill space.

grid-auto-flow: column; helps auto place items column by column.

**TERMINOLOGY**

hero class is basically called hero because it is the section that is placed prominently, similar to banner.