CSS – cascading style sheet

Can’t write CSS can't work without HTML

Remember to inspect the element of other website in order to gain inspiration from other websites

CSS is usually listed under style attributes

Ways to put in CSS

* Using <link> to link the external css file and can be used on multiple html
* Make sure to break them down to not have one huge css
  + Same idea to not have too much css files because it defeats the purpose of organization. It is not organization to have each attribute having one css file to them
* Can also use IMPORT
  + Import takes external css from the web and can be imported into your website
  + Commonly used for .sass, where multiple .sass are compacted before importing in as a .css file

**Inline CSS**

Using p, h1, etc, we can put css directly into the element within html.

This is not preferred because it does get more complicated the more elements you have and would be harder to keep track between all elements.

**Internal CSS**

Puts css in the head between a <style> tag, it's better than inline CSS but it only applies to that one page and would need to be copied over to every other page that needs it. It is best to have this when only doing it for one page entirely.

Consider the images that would be placed on the screen. Most images taken from professional cameras or even cellphones would produce a jpg file image

* Has high retina density and weighs on the website to load it
* **Rule of thumb, resize the image 2x the size of the thumbnail** so that image does not need to be loaded with the same capacity it needs to load in the full image
  + For example, code: set thumbnail images to 600px width || background image at 1400 to 2000px wide

Make sure that css is linked **relative** to the file path instead of an **absolute** path.

Note: changing the width by default will scale the height with it until it gets a height attribute for itself.

Using **absolute paths**, it links to other websites but be mindful when doing it for IMAGES

If anything happens to that image (relocating, editing) the image may go away

Also, it is not fair on the linked side because you are taking their bandwidth to access that image for your own website. Try to have a link direct to the hosted image you want, if not within the files then have it stored in a safe place outside of your files

There is the W3C documentation has all the rules of web specification for different paths such as html, css, etc.

* **Recommendations are the newest update documentation**

Css syntax

* Takes in a selector, writing declarations within a declaration block
* Selector example: img
* Declaration: width: 300px;

There are short hands and long hand forms for properties

* Example: padding has 4 different properties for all 4 sides of a box
* Long hand = padding-top, padding-right, padding-bottom, padding-left (clockwise)
* Short hand = padding: top right bottom left

Css **comments** = /\* \*/

* Used to leave notes for yourself
* Organization

Css organization guides can be found here: https://codeguide.co/#css-declaration-order

**Length**: specifies sizing

* Absolute: fixed unit, always the same size
* Relative: relative unit depend on parent/descendent elements

--- Chapter 2

There are function values within css depending on the declaration you use

* **Transform** takes in rotate(degree)
* **Width calc** (compute math values)
* **Background image** (url)

Default browser/website style

* Initial values that would be a backset style before adding any additional css
* Ex: h1 is defaulted bold
* It can be changed with css overwriting that attribute \

Color property:

* Have basic color keywords and Extended keywords
* Basic colors: red, blue, green, cyan
* Extended color keywords: aliceblue, mediumseagreen

Use: <https://colours.neilorangepeel.com/>

Can use Hex codes with the RGB values

* #fff = #ffffff
* rgb(0,0,0)
* rgba(0,0,0,0) a = alpha component = opacity
* hsl (degree, %dark, %light) = hue that takes in a hue
* Hsla() includes alpha/opacity

CSS SELECTORS

1. **Type** selector: selects a particular type ( h1, img, p, etc )
2. **Universal** selector: \* (uncommon because it is not as common to want to change everything to have the same css)
3. **Class** selector: selects a particular class with a .
4. **Id** selector: selects a particular class with a #

Note that id can only be used once per page (as an id)

Naming convention: make sure it is meaningful and understood

When using selectors the selectors goes as:

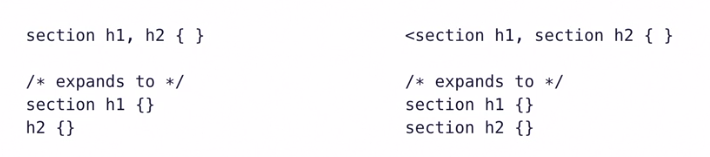
**class/id** > **type** > **types** (if it was a link or something within the type selector of that class/id) > **action** (hover, clicked, etc)

Note that the css will take the highest level as the style it takes in.

If you have a [ .class a {} ] declar block and a [ .class p a {} ]

* It will take the first one as the css tag before the second \*with the p tag\* because it has higher importance. To avoid this, make sure to take away **overlapping** attributes

Group selectors can be used by adding commas



---

Inheritance: css styles can be inherited from the ancestor to descendant elements.

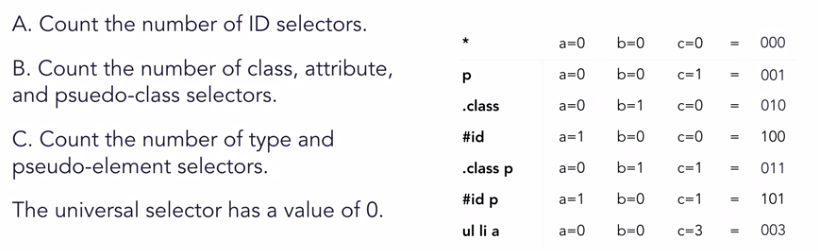
Ex: applying style to the body tag, every element within that body tag will also get that style

Note: not all styles can be inherited

Hierarchy of selectors (the highest in the hierarchy will override anything under it)

Id > class > type > universal

1. Universal (\*)
2. Type (p,h1,section,etc)
3. Class (.example)
4. Id (#example)

Calculations for the hierarchy:  


The highest number is the one that shows.

Cascade: refers to how style rules are applied based on specificity and source order.

* The latest change will be the one that gets shown
* The order of the style sheet Is read top to bottom
* Using !important will place importance to style but it is commonly frowned upon since it obstructs the hierarchy of css

Pseudo selectors: selectors that requires interaction with the webpage to activate

These examples are associated with links:

* :**link**
* :**visited** – when the link has been clicked on before
* :**hover** –hovering with a mouse over an element
* :**active** - when it is being activated
* :**focus** - element when trying to navigate using find on a webpage (tabbing through)

Inline | block | display

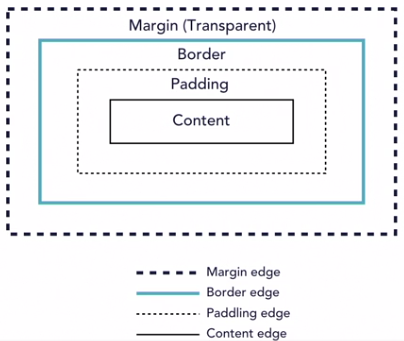
* **Inline**: same line, starts from left and piles to the right, and will wrap around once it reaches the edge to the right of the window
* **Block**: stacks on top of each other, same height as content | same width as container, always starts a new line
* **Display**: used to change the default behavior of inline and block elements (move inline to block, move block to inline)

Note:

1. Adding height to an inline element will not work

---chapter 3

Box model:



* **Margin**: adds space around the box outside
  + -auto ,-negative, (directional)
* **Border**: the outline of the element
  + -width, -color, -style
* **Padding**: add/remove space within the borders
  + Can't have negative, (directional)
* **Content**: the properties added within

The box model will always be in the inspector tool, where you can inspect the amount of padding/border/margin/content and the number of units inside

NOTE:

1. there are some html properties that have default padding/margins, to get rid of it in general and then fix it again for yourself later, set margin and padding to 0 in the global css attributes
2. MARGIN COLLAPSING: margins between each element does not stack,   
   if there were **margin-bottom: 10px** on one element and **margin-top:10px** on the element right under it, there will only be a space of 10px.

Margin and negative values:

By default, block elements stack on top of each other.

In order to have block elements to stack on top of each other, you would need a negative margin value to move the elements outside of the stacking position.



**Margin can also help with centering elements using: margin: 0 auto;**

**Margin: (top and bottom = 0) (left and right = auto --> which means distribute evenly)**

Values and Units: for box properties, the <percentage> and <length> data types are used

* Width is considering the container area.

Relative <length> units

* **Px**: pixel
* **Em**: inherited font-size of element
* **Rem**: font-size of the root element
* **1vh**: 1% width of the viewport
* **1vw**: 1% height of the viewport
* **Vmin**: equal to the smaller of vw and vh
* **Vmax**: equal to the larger of vw and vh

---Chapter 4

**Typography**

Arranging type for readability and to engage and communicate with the readers.

**Typeface**  
A set of fonts designed with common characteristics, composed of glyphs.

Different typography for different meaning

* Times new roman: legal, official papers
* Comic sans: casual and fun

Types of typefaces

* **Script** (small decorative detail)
* **Decorative** (heading/decorative details)
* **Monospace** (usually used to display code)
* **Serif** (common – more formal)
* **Sanserif** (common – more modern)

**Font-family**: how to add typefaces into the webpage

Make sure that there are generic font family name just in case the font is not available.

(generic) then (specific)

**Web-safe fonts:**

Fonts that are commonly pre-installed on most computers or devices (arial, times new roman, etc)

Fonts may vary among operating systems.

Font-weight: the thickness of the font size (lower = thinner, higher = thicker)

* Normal = 400, default for body text
* Bold = 700, default to headings

Font-style

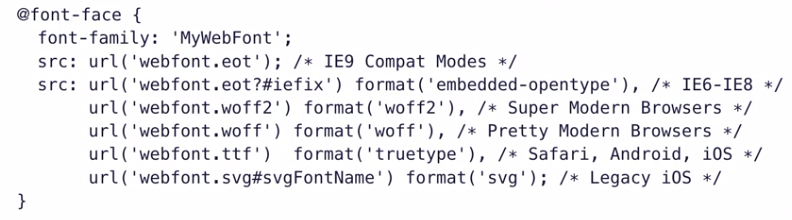
Used to add or remove an italic style. (italic, oblique, normal). It is different between each font, so keep that in mind.

Web fonts can take fonts from the web to use in your webpage

1. Web font must be linked in the html file
2. Call on the font in css

The safest way is to include the font .ttf within the project folders under “fonts”, so that the relative link to it will always be there and there is no fear of it changing

You are also able to url link your fonts in the font family, but make sure to know what the browser compatibility is to that font.



Absolute values are not dependent on anything

* px

Relative values are based on nearest ancestor elements

* em (named after m) -decimal points (1em = inherited font size)
* rem (root em)
  + Only relative to html

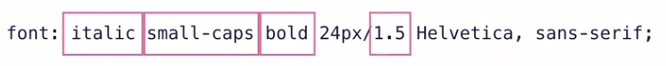
+ note that relative values are relative to the % of the parent container

This is usually used for resizing applications for different screen sizing

There are longhand and shorthand for declaring font family.

The generic rule for short handing font families are:

Font: [font-style] [font-varient] [font-weight] [line-height]



Note: Order matters!!

Text-decoration: [line] [color] [style]

* This is the attribute used to get rid of default lines in links
* Also can provide decorations to normal text

Text-align used to align content within a block element

Line-height sets the height of the space between two lines of text

Closely related to font size and can use different types (px, %,ems,rems,unitless)

--- Chapter 5

Floats

\*\*may not be supported with internet explorers, but there are other ways to have float

Float allows images to be enwrapped with text, like a magazine

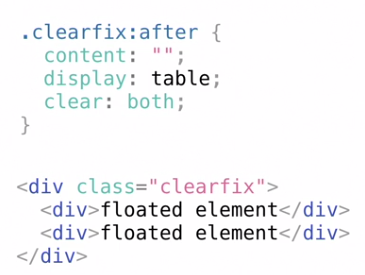
With float, it will take the element off the flow of the container and the normal flow will be “floating”

Parent elements of the float content will only wrap around non-floated children.

Overflow: used to show how to display content that doesn’t fit in its containers

* Hidden can be used to clip out overflowed content
* Auto give a scroll bar if needed

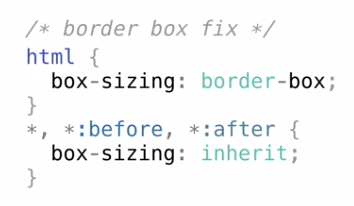
Clear fix hack: added to the parent of floated elements



Floats can be used in order to organize a webpage box layout

**Box-sizing:** content box vs box-sizing border-box;

* Using box-sizing border-box, it will allow you to include the padding and border space within the dimension without affecting the space of the content within BUT it will make sure that the overall box size stays the same.



Positions: can also help move elements around. Has all directional properties

* **Static**: not positioned
* **Relative**: relation to current position
  + Adding any top/right/bottom/left value will fix it from it’s 0,0 position of that ‘current’ container
* **Absolute**: relative to contained element
  + Places the content in a new container of its own and ignores other flow of content.
* **Fixed**: relative to the viewport
  + Stays in the same spot even with scrollbar
* **Sticky**: relative to containing element and viewport
  + Experimental
  + Stays in the initial spot until scroll bar hits it, then it turns fixed

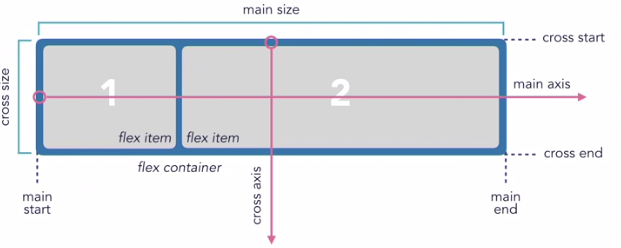
Stacking context: **z-index**

Can show what is behind and forward depending on the z-index. Think of it like layers of paper on top of each other

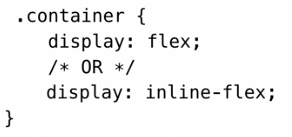
Flexbox / grid

Flexbox = 1 dimensions

* Distribution across a single axis



To use flexbox, it must be defined that the container is a flex property



Note when having inline flex: it is not affecting the content inside to be inline, but the container itself to be in line to the next container

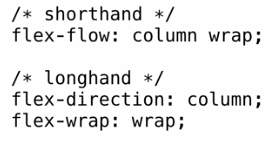
Flex-**direction**: determines the direction of the main axis \*\*note most flex has a reverse function

There are four values

* Row: will have it in a line stacking from left to right (can be –reverse)
* Column: will have it stack top to bottom (can also be –reverse)

Flex-**wrap**: no wrap – will make it so the container will not wrap against the content inside

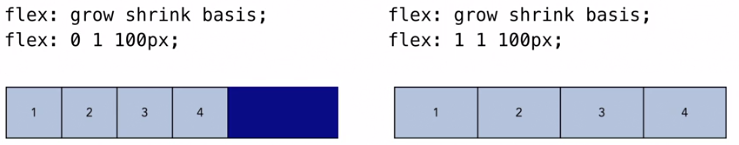
* With wrap, there would be a take the container size and push the other items to the new line.

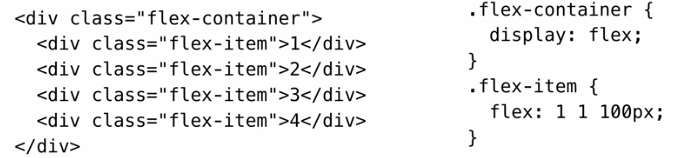


**Flex** properties

* Flex-**basis**: sets the initial size of the flex-items
* Flex-**grow**: determines how items will expand if there is extra space in the container
* Flex-**shrink**: determines how items will shrink if there isnt enough space in the container

This all can be done in short hand with the order being:



Note that in order to use flex properties, it needs to be linked from the html class to CSS  


**Justify-content:** aligns items on the main axis

* Has start, center, space-between, space-evenly, etc

**Align-items:** aligns items on the cross axis

* Has stretch, center, start, end, etc.

**Grid** = 2 dimension

* Layouts with both rows and columns

Grid container – parent element of grids

Grid items – the child elements within he grid container

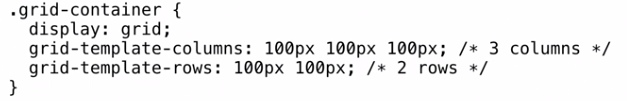
Grid gutters are the space between reach box within a grid

Grid tracks are references to the columns or rows of a grid

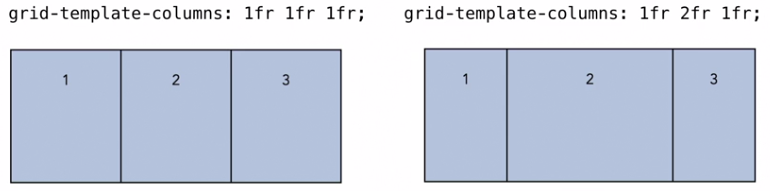
**To make a grid:**

1. **Grid-template-columns**
2. **Grid-template-rows**

This is defined within the css like:



Fr: represents a fraction of the available space in the grid container



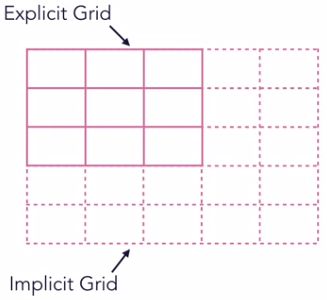
**Repeat([number of tracks], [size of tracks])** allows you to repeat tracks without writing it over and over

Ex: grid-template-columns: repeat(3,1fr);

The original grid layout specification includes the grid-gap, grid-row-gap, and grid-column-gap properties.

* Has been updated to gap to be used for both grid and flexbox

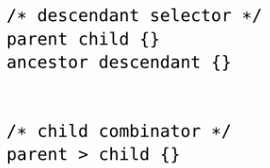
The grids made above are all explicit grids: we set the layout, but we are able to do auto columns and auto rows in order to set the stage for the rest of the grid.



---- Advanced selector

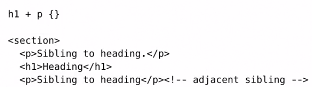
Descendant Selectors: Creates matching patterns based on the relationship between nested elements

**Child** combinator (>): Only matches to direct child elements

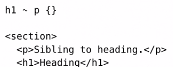


Sibling combinator (+, ~)

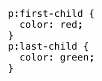
**Adjacent** Sibling combinators (+)



**General** siblings combinators (~)



Selecting **first** child & **last** child: Selects the **first** instance and/or the **last** child of that particular type



----- Responsive Design

Responsive Web Design (RWD)

1. Fluid Layout
2. Flexible Images
3. Media Queries

Use max-width / min-width: limits how the windows react to different size window

----- flexible background images

Images: images that are part of the content should be added with HTML

<Img src=”images/image.jpg” alt=” course thumbnail”>

Images that are presentational should be added with CSS:

Section {

Background-images: url(path/to/images.jpg);

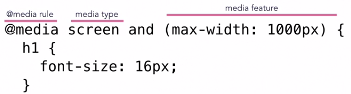
}

Background properties:

* Background-size: cover – fills in the container
* Background-repeat: no-repeat – won't have image repeat
* Background-position: top right – put image to a certain area

Media Queries

Depending on a media screen, it will change CSS layouts in a certain way



Media Types

* All: matches all devices (default)
* Print: patches to printers / print-related displays
* Speech: matches to screen reading devices
* Screen: matches all devices that aren't categorized as print or speech

Can use import to load the stylesheet into the main CSS file

Media Query – Breakpoints

Break points are for particular screens for the computer to recognize

* Mobile portrait
* Mobile landscape
* Tablet landscape
* Tablet portrait
* Anything bigger than tablet --> desktop

@media (width: some\_size) {… }

Min and max width breakpoints: have a target for both min and max in order to make a range.

1. Desktop   
   With desktop as a base model, there are more often max-width in order to make sure that all content is displayed
2. Mobile first  
   With mobile as base, there is more min-width

Test this by resizing your window and on your own phone

**References**

General:

<https://css-tricks.com/>

<https://www.w3schools.com/>

<https://developer.mozilla.org/en-US/>

<https://caniuse.com/>

Images:

<https://stocksnap.io/>

<https://burst.shopify.com/>

<https://unsplash.com/>

Colors:

<https://colours.neilorangepeel.com/>

<https://coolors.co/palettes/trending>

Fonts:

<https://fonts.google.com/>