**CSS** (Cascading Style Sheets)

Ci consente di dare uno stile alle nostre pagine web.

Css zen garden--> sito con una serie di possibili stili html e css.

Ci permette di rendere le nostre pagine responsive, creare animazioni, ecc..

Ci permette di favorire la separazione tra lo stile di un documento e il suo contenuto. (concetto di Saparation of control, separazione tra contunuto, stile e intrazione).

Ci permette di rsparmiare tempo scrivendo una sola volta il codice css che sarà poi applicato su molteplici elementi.

1. Lo stile css si aggiunge a un file html. Ci sono 3 modi per fare ciò:
2. Un stile esterno con un tag **<link>** che indica un file esterno da inserire nel tag head dell’html. È ideale quando si intende applicare i medesimi stili a diverse pagine dello stesso sito. È possibile agganciare un foglio di stile esterno in 2 modi: linkando o importando. Se si linka il file dovrà avere estensione .css. Questo è il modo più giusto per inserire stile css in un documento html.
3. Uno stile interno con tag **<style>** sempre nella head del documento. Agisce solo sul documento nel quale viene inserito. Dovremmo evitare di usare questo tipo di scrittura css per il principio di separazione, in quanto usando il tag<style> nella head di un documento html non si separa lo stile dal contenuto. Spesso è consentito però l’uso dello stile interno in 2 circostanze:
4. Quando vogliamo fare dei test rapidi su una pagina possiamo scriverlo nel tag <style> nella head e successivamente spostarlo nel file css esterno una volta che siamo certi del risultato.
5. Quando siamo interessati a stilizzare in maniera più specifica degli elementi che sono stati già stilizzati dal css esterno.
6. Uno stile in linea usando l’attributo style nel tag di apertura di un documento html qualsiasi. Viene usato per dare uno stile unico a un elemento scrivendo del codice css direttamente all’interno del elemento html stesso. L’attributo style include una serie di coppie proprietà valore. Ogni coppia è separata dalle altre con un punto e virgola. È generalmente considerata una pessima idea usare questo tipo di scrittura css perchè: contrasta con il concetto di Separation of control e abbassa fortemente il punteggio SEO della nostra pagina in quanto va contro il principio DRY (don’t repeat yourself), ovvero se volessimo che tutti i nostri tag <p> fossero di un determinato colore dovremmo scriverlo in ogni singolo tag <p> e se volessimo cambiare colore dovremmo cambiarlo a mano su ogni singolo elemento.

Concetto di specificità-->

Una regola css si compone di 2 parti:selettore+Una o più dichiarazioni

Un selettore --> specifica a quale elemento/i intendiamo applicare lo stile definito tra le parentesi graffe. Ogni coppia proprietà valore costituisce una dichiarazione--> la proprietà è l’attributo di stile che si vuole impostare. Può essere quasi qualsiasi cosa (font, colore, background, ecc..). Il valore può avere sintassi diverse tra proprietà e proprietà e sono possibili moltissimi valori per la medesima proprietà. Possiamo dividere i selettori css in 5 categorie

1. Selettori semplici
2. Selettori di attributi
3. Combinazioni
4. Pseudoclassi
5. Pseudoelementi

I principali sono:

- \*--> selettore universale, seleziona qualunque elemento sulla pagina. Molto debole come selettore e viene usato solo quando bisogna dare a tutti gli elementi il medesimo font.

-element type--> seleziona tutti gli elementi con il tag name indicato, esempio p {}.

-id--> seleziona un elemento singolo, unico presente come valore di un attributo id in un tag html.

-selettore classe--> può essere utilizzato per selezionare qualunque elemento html con un attributo class con uno specifico valore indicato. Possiamo concatenarlo con un attributo element type.

-descendant selector-> permette di selezionare un elemento che è discendente di un altro elemento. Es. p span { color: blue; } colora di blue tutti i tag span dentro un tag p. Anche se nel mezzo ci sono altri tag.

-child selector--> per selezionare gli elementi selezionati a destra (figli diretti di un genitore).Es: div > span {} deve essere per forza figlio diretto del div, non ci possono essere di mezzo altri tag.

-sibling selector-->ci permettono di assegnare degli stili a partire da un elemento.

Es. div +p {} ovvero il p subito dopo prende la formattazione.

-sibling general selector--> div ~ p {} anche se non è subito dopo ma c’è di mezzo un tag

(la tilde si fa alt+ 1,2,6)

DIFFERENZA TRA SIBLING GENERAL SELECTOR E DESCENDANT SELECTOR?????????

Ci sono 3 meccanismi che determinano quale regola dovrà applicarsi in caso di conflitto:

1. CONCETTO DI CASCATA--> a parità di altre condizioni i fogli di stile si applicano a cascata, ovvero se esistono due regole uguali si applica quella più vicina all’elemento. Maggiore rilevanza in line, poi interno e poi esterno.
2. CONCETTO DI SPECIFICITÀ--> quanto il selettore css va in modo specifico a targhettizzare un elemento specifico del nostro documento. La specificità vince sul concetto di cascata. “Css specifishity” sito per determinare le specificità. !important massimo livello di specificità, maggiore anche di stile in line. Meglio non utilizzaro mai, ma ha senso usarlo quando siamo a lavorare con un framework, come bootstrap.
3. CONCETTO DI EREDITARIETÀ-->alcune proprietà css vengono ereditate, quindi se assegniamo una proprietà a un elemento e al suo interno questo ha altri elementi queste proprietà possono essere ereditate. Es: dare un colore al body trasferisce i colori a tutti i p al suo interno. Però non tutti gli elementi, come per esempio il tag encore.

PROPRIETÀ CSS

Proprietà--> ci permettono di reinderizzare il testo in modi diversi:

* **Font-family**--> indicare al browser quale deve essere il font da utilizzare. È possibile indicare più font nel caso il browser utilizzato dall’utente non supporta il primo indicato. I font più leggibili dall’utente sono serif e sans-serif (font generali). Monospace imita il carattere di una macchina da scrivere (utile per mostrare del codice). Quando il font è composto da più parole bisogna scriverlo tra apici.
* **Font-style**--> modifica lo stile del font scelto.
* **Font-size**--> modificare le dimensioni del font. Il modo più rapido è assegnare un valore in pixel, ma sono accettati anche dichiarazioni in em (si riferisce al padre,es. Padre=16px, figlio=1em(=16px)), rem (si riferisce al root (solitamente html), % (per il font, relative al genitore. Per il margin e padding si riferisce alla width del parent. Per height e width si riferisce al parent). Scarsamente utilizzate sono le keywords small, medium e large, smaller, larger.
* **Font-weight**--> spessore del font, possiamo specificare normal, bold, bolder, lighter oppure possiamo indicare un numero.

Per trovare l font migliore si può utilizzare google fonts, il quale ci permette di selezionare un font tra quelli indicati (es:orbitron), ti fa copiare il tag link che bisogna mettere all’interno del codice html. Per scegliere dei buoni font si utilizza di solito solo dei font sans-serif (senza grazie) ma per degli h1 si possono utilizzare dei font con grazie o dei font display. Mai più di 2 font. In fondo ti suggerisce anche gli accoppiamenti di font.

Per inserire una grande quantità di **colori** ci sono diversi modi:

* Modo esadecimale in base 16, cancelletto iniziale+6 cifre esadecimali: #36fa21, questo è pari a una gradazione di verde. Queste 6 cifre conviene pensarle divise a blocchi di 2, le prime due si riferiscono al canale del rosso, le seconde due al verde, le ultime due al blu. Quindi volessi massimizzare il canale del rosso dovrei fare #ff0000 (uguale a red)
* Con sintassi rgb che divide un colore nei tre canali rosso, verde e blu. Ogni classe di colore arriva massimo a 255, di conseguenza volendo un rosso pieno bisognerebbe fare rgb (255,0,0)
* Sintassi hsl (hue saturation lightness) i valori vanno per la tinta da 0 a 360 su quella che si chiama color wheel, per saturazione e luminosità da 0 a 100 %.

Un'estensione utile per inserire i colori è colorize.

* Per l’opacità dei colori invece si usa la sintassi rgba che è come l’rgb ma ci permette di aggiungere un canale alpha. Es rgba(255,0,0,1) così è un rosso opaco al 100%, volessi 50% si mette 0.5.
* Per mettere dei colori background si usa la proprietà  **background-color**. Per mettere delle immagini **background-image.** Per fare o non fare ripetere l’immagine nel background allora bisogna usare **background-repeat.** Per cambiarne la posizione **background-position.** Per determinare quanto questa immagine si deve estendere usiamo **background-size.** Questo ha keyword contein e cover,che ci permettono di estendere l’immagine, per contain sarà ingrandita fino a quando non incontra i bordi dell’elemento per il quale è stata assegnata come background, per cover coprirà tutto l’elemento tagliando anche parti dell’immagine. Per dare effetto di parallasse **background-attachment.** Se oltre all’immagine vogliamo per esempio aggiungere colori, non usiamo più background-image ma solo **background** con oltre all’immagine il comando linear-gradient, si possono mettere i gradi per dara una direzione alla colorazione, es 120deg. Proprietà :hover dopo la classe permette di cambiare il colore del background passandoci sopra con il mouse.
* Per posizionare degli elementi usiamo **position.** Static, fermo nel documento, si usa poco. Relative determina la posizione relativa nel flusso del documento html, ora si può muovere con top, bottom, right, left. Significa “spostamelo di tot a partire da top, ecc... Position absolute invece si usa in relazione con il padre che deve essere già posizionato. Quindi il figlio con posizione absolute si muoverà all’interno del padre. Quindi se il padre non ha una posizione l’elemento con position absolute si riferirà al body, ovvero l’unico suo contenitore con position definita. Position fixed fisserà un elemento in maniera indipendente rispetto alla pagina (es: tasto scroll to top nei siti). Position sticky fa cambiare la posizione da relative a fixed quando incontra la condizione prestabilita, es: top: 120px; (utilizzabile per navbar).
* Per **sovrapposizioni** si può usare un combinatore per selettori-->

.class1, .class2 {} con position relative, poi volendo spostare la class1 di un tot di px si fa .class1 {left:50px;} . Sta sopra l’elemento inserito per ultimo nel documento, ma si può sovrascrivere anche questo mettendo a class1 la proprietà **z-index**, con un valore che deve essere maggiore rispetto a quello di class2 (che non essendo specificato è 0). Funziona solo su elementi posizionati.

* **Box-model**: se si da il background-color senza specificare, ci colora soltanto il content e il padding. Ci sono più modi per assegnare dei valori alle parti della box. Se si assegna per esempio 20px al **padding** significa dare 20 px di imbottitura all’area di content per ogni lato (che aggiunge, quindi si ingrandisce la box). Si possono dare anche differentemente per ogni lato *partendo dall’angolo in alto a sinistra in verso orario*. Padding: 0px 15px 0px 15 px; se si fornisce solo due valori il primo si riferisce a top, bottom, il second a left, right. Se ne specificano 3, il primo a top, il secondo left, right, il terzo a bottom. Queste proprietà sono associabili anche al **margin**. Non si usa il margin per posizionare gli elementi, ma con position. **Border** --> solitamente si assegna un bordo uguale per tutti i lati. Si differenzia dimensione, stile, colore. Border: 4px solid blueviolet. (dashed--> tratteggiato). per rendere i bordi arrotondati si usa **border-radius**--> sia px che %, si danno le misure dall’alto sinistra in poi.
* **Display**--> proprietà es: span ha prop. Inline mentre il div ha block(che è di default). La prop display con valore in line azzera le grandezze di height e width (perché inline fa si che le dimensioni siano date dal contenuto, es. Un<p>) ma con inline-block non succede, es: con due box vuote mettendole in-line scompaiono. Con la proprietà flex, a differenza dalle altre proprietà, display, non va data direttamente agli elementi su cui vogliamo agire ma al contenitore, l’elemento padre. Di default i figli si dispongono in line anche senza contenuto. Di default vengono messi a sinistra. Con proprietà **justify-content**, di valore predefinito ha start, ma si può metterespace-around, agisce sullo spazio orizzontale che c’è tra i figli e tutto lo spazio che c’è a disposizione lo mette attorno ai figli, quindi li dispone a distanze uguali(simmetrico) stesso spazio tra elemento e bordo ed elemento e centro. Space-between mette lo spazio in mezzo, uno a sinistra e uno a destra. Space-evenly mette lo stesso spazio tra gli elementi e i bordi. **Align-items**  agisce sui blocchi ma in verticale, se mettiamo proprietà baseline*,* quest’ultimo allinea gli elementi indipendentemente dalla grandezza. **Flex-direction** default è row(orizzontale) ma se mettiamo column gli assi vengono invertiti quindi l’asse verticale diventa l’asse x . **Flex-wrap**, che di default ha no-wrap, che pur di mettere gli elementi nello spazio orizzontale modifica le dimensioni dei figli. Con wrap lascia le dimensioni che abbiamo dato e manda a capo anche superando le dimensioni del padre. **Align-content**center*,* end*,* startgestisce tutto il contenuto.
* Per rendere il sito *responsive* deve essere adattabile al tipo di device (computer, tablet, mobile). Ciò si fa attraverso le **media-query**. Ovvero degli strumenti che ci dicono come il sito si deve comportare quando incontra un *break-point*. Cercando “breakpoints media-query css” escono le grandezze di riferimento per i vari device. Si inserisce sul css facendo: **@media screen and (condizione da verificare)**. es:@media screen and (min-width:600px) { .box {background-color: red}}. la condizione della classe box avviene solo se il device ha minimo 600px. Ovviamente si può mettere anche max-width. Si possono concatenare più dimensioni: @media screen and (min-width:600px) and (max-width:900px) {}.
* **Animazioni--> Transforms**, dobbiamo agire tenendo in considerazione i tre assi (x,y,z) può avere come valore scale che mette in scala per i valori dati-- > *scale (2,3*); poi c’è skew che ci permette di allargare l’elemento su due angoli opposti, es: *skew (20deg). T*rasform-origin*-->* da dove parte la trasformazione. Il più utilizzato è translate-->trasliamo sui 3 assi es. Translate (x,y,z). Poi abbiamo **animation-name** che serve per dareun nome all’animazione. **Animation-duration**, per dare una durata all’animazione. **Animation-delay**, per dare delay all’animazione (dopo quanto si vuole che parta). **Animation-iteration-count**, per quante volte vogliamo che l’animazione si svolga. **Animation-direction**, serve per dare la direzione, come valore possiamo mettere alternate, normal, alternate-reverse. **Animation-fill-mode**, che con valore forwards blocca l’animazione al 100%.

Dentro la classe si scrive:

.nomeclasse:hover{

Transform:(1.2);

}

Dopo aver dato le caratteristiche dall’animazione bisogna fare--> Es:

**@keyframes “nomeanimazione” {**

**0% { transform:translate:0px;)}**

**50% {background-color:blue;}**

**100%{transform: translateX (200px);}**

**}**

* Per memorizzare, principalmente i colori nel foglio css creiamo un contenitore che si chiama **variabile** (che deve essere subito riconoscibile per non confonderlo con altri contenitori).

**:root{**

**--purple: rgba (valori);**

**}**

Quando vogliamo usarlo usiamo la keyword **war(nome variable)**, es: war (--purple).