

Web

- HTML
- CSS
- JavaScript
- Esempio
 - <https://github.com/egalli64/mswat>
 - Maven
 - Tomcat 9

mswat: applicazione che corre dentro tomcat

application server, genera pagine e verifica che l'utente abbia il diritto di accedere a esse

Tre tecnologie per il web

HTML struttura, **CSS** stile, **JavaScript** interattività

- Standard del W3C

<https://www.w3.org/standards/webdesign/>

- MDN Mozilla Developer Network

<https://developer.mozilla.org/it/docs/Web>

per documentazione ufficiale su sviluppo web

Sviluppo su Eclipse via Maven

- Apache Tomcat: <http://tomcat.apache.org/>
- maven-archetype-webapp
- pom.xml
 - Properties
 - Specificare la versione di Java e il source encoding
 - Dependency
 - groupId: javax.servlet, artifactId: javax.servlet-api, version: 4.0.1, scope: provided
- Il codice sorgente va in src/main/webapp
- Run on Server mswat corre su tomcat
 - Target runtime: Server view (Window → Show View → Servers)

hyper text mark-up language per potersi passare documenti attraverso dei link. nasce dalla necessità, al CERN, di indicizzare e scambiarsi documenti.

HTML: HyperText Markup Language

linguaggio di mark-up che mi permette di descrivere una pagina. è un linguaggio descrittivo, mi fa il rendering di una pagina (la fa vedere in un certo modo)

- Tim Berners-Lee @CERN ~1990
- World Wide Web Consortium (W3C) **HTML5** 2014
- Descrive come rappresentare pagine web
il compito del programmatore web è di verificare che il codice web che hanno scritto sia renderizzato nel modo voluto
- Il rendering è responsabilità del browser
 - Chrome
 - Firefox
 - Safari
 - ...
- Struttura ad albero, ogni nodo è un elemento
 - **DOM**: Document Object Model
cugino del POM

documento xml con struttura ad albero

```
<!doctype html>
<!-- my hello page -->
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

radice dell'albero

hello.html

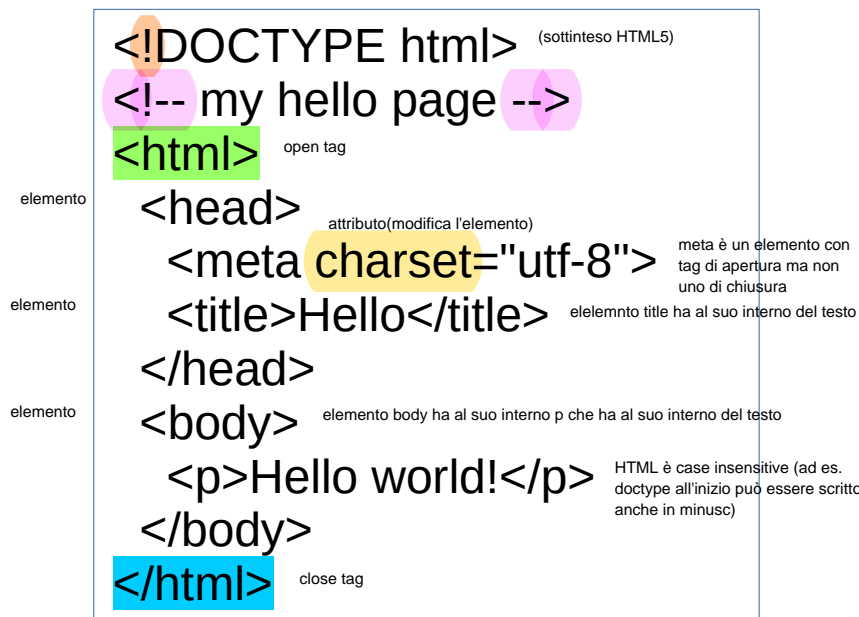
Elemento

Singolo componente di un documento HTML

- Normalmente delimitato da **open** – **close tag**
 - In alternativa, elementi vuoti non hanno il close tag
 - I tag sono case-insensitive
- Può contenere testo e altri elementi
- Può avere **attributi** nella forma *nome="valore"*
 - Attributi booleani nella forma *nome* o *nome="nome"*
- “!” indica che **non è un elemento**
 - **DOCTYPE** tipo di documento. Aiuta il browser a interpretare correttamente il codice (qui: HTML5)
 - **Commenti HTML**: `<!-- ... -->`

in HTML apice singolo e doppio sono uguali

elemento HTML (terminologia del programmatore web)=corrisponde al nodo di un albero



questo viene generato quando il browser legge una pagina

<x attr> è un attributo booleano che ritorna true (perchè non c'è un "uguale a qualcosa")
attr = "attr" (abbastanza raro) attributo booleano che ritorna true

un documento HTML deve avere sempre un head e un body al suo interno. il body sono le informazioni che io voglio dare. head è la parte di intestazione con le meta informazioni sulle informazioni che voglio dare all'utente. la head interessa al browser perchè gli dice che, ad esempio, usa caratteri in UTF8. quando il programmatore web dimentica di mettere questa informazione, il browser mette la tabella di conversione che pensa sia adatta (quindi ad esempio compaiono caratteri strani
i programmatori web tendono ad usare il termine tag per parlare di un elemento

head vs body

- html

- Contiene l'intero codice HTML della pagina

metainformazioni (utili al browser)

- head

- Informazioni *sulla* pagina

informazioni visibili all'utente

- body

- Informazioni *nella* pagina
 - Contenuto che vogliamo mostrare all'utente

```
<!DOCTYPE html>
<!-- my hello page -->
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

head

- Gli elementi in head hanno lo scopo di descrivere la pagina corrente
 - **title**: il titolo della pagina, solitamente mostrato dal browser nella barra del titolo
`<title>Hello</title>`
 - **meta**: informazioni aggiuntive, come l'encoding usato e indicazioni per i motori di ricerca

(meta contenuti
nell'elemento head)

```
<meta charset="utf-8">
```

descrive cosa ci si aspetta dalla pagina

```
<meta name="description" content="Writing HTML code">
```

"se qualcuno ricerca una pagina in cui si spiega come scrivere un HTML code fai salire la mia pagina in classifica perchè li parlo di quello"

```
<meta name="keywords" content="html, head, title, meta">
```

questo è ambito SEO (metti questo meta per fare in modo che la pagina compaia per prima)

- Per altri tipi di metadati vedi
 - The Open Graph protocol <https://ogp.me/> fornisce dei meta che possono essere usati dai motori di ricerca

Testo

- **h1..h6** rappresentano i titoli (Header1 per il titolo principale). usati per dare una struttura al documento
 - Titoli (heading) di parti del testo, di solito ci si aspetta un solo H1 per pagina
- **p** ogni paragrafo è delimitato da: <p> e </p>. al loro interno, se scrivo lasciando tanti spazi e andando a capo ma rimango sempre nei tag del paragrafo, il rendering rende il testo normale senza spazi nè a capo. per andare a capo devo scrivere

 - Paragrafo, unità di base per la suddivisione del testo
- **b, i, u, sup, sub, ...** formattazioni di base (agli scienziati che crearono HTML servivano solo queste, ora per i siti web si usa CSS per avere formattazioni più complesse (se ne occupa lo UX designer)
 - Formattazione del testo, (bold → grassetto), <i>(italic → corsivo)</i>, sottolineato, esponente, pedice, ...
 - Obsoleti (andrebbe usato CSS) ma mantenuti per compatibilità e semplicità
- **em, strong**
 - Enfasi, importante
- **br**
 - HTML ignora molteplicità di spazi, tab, andate a capo, etc. Ogni gruppo è interpretato come un singolo spazio bianco.
 - Per forzare l'andata a capo si usa
 o
, elemento che non ha tag di chiusura
- **hr**
 - Per separare blocchi nella pagina si può usare un horizontal ruler <hr> o <hr/>
horizontal ruler (ora si usa solo quello senza slash)
questo non ha tag di chiusura

Caratteri speciali

- Alcuni caratteri non utilizzabili in HTML, o non disponibili su normali tastiere, sono resi con “entity”, stringhe che iniziano con “&” e finiscono con “;”

< <

€ €

> >

¢ ¢

& &

© ©

" "

® ®

- <https://dev.w3.org/html5/html-author/charref>

Liste

- **ol** ordered list (lista con numeri)
 - Lista ordinata in cui ogni voce ha un indice crescente
 - L'elemento ol contiene un elemento **li** (list item) per ogni voce
- **ul** unordered list (lista con pallino, quadratino, ecc)
 - Lista senza ordine, come ol ogni voce è un **li**, ma pallino (o altro) invece di indice
- **dl** (secondarie)
 - Lista di definizioni, dl può contenere ogni combinazione di elementi **dt** e **dd**
 - dt (definition term), il termine da definire
 - dd (definition of definition), la definizione del termine
- Una lista può contenere altre liste
 - Ci si aspetta comunque che un elemento LI inizi con testo, e poi segua l'eventuale lista

```
<ul>  
<li> ... </li>  
<li> ... </li>  
</ul>
```

Link

Gestione dell'ipertestualità nelle pagine HTML

anchor

hypertext reference: c'è indirizzo risorsa a cui voglio andare quando clicco sul link

- a – href
 - anchor to an hypertext reference, “ancora” l'elemento ad una **risorsa** definita nel suo attributo href
 - all'interno del tag di apertura c'è una risorsa, quello che vede l'utente è l'index page scritto in blu sottolineato col link
 - risorsa interna: `index page`
 - elemento nella pagina corrente
 - attributo "top" di tipo id
 - Definito un elemento con un dato id: `<h1 id="top">Hello</h1>`
 - Un anchor può linkarlo così: `the top`
 - href a (un elemento in) un risorsa nel web: `https://www.w3.org/#w3c_crumbs`
 - mail-to: `site administrator`
- l'attributo title può essere utilizzato per dare informazioni aggiuntive al link

Immagini

img è lo spazio nell'HTML in cui voglio che venga visualizzata l'immagine, non è quindi l'immagine stessa

elemento fatto per contenere delle immagini da visualizzare nella pagina web

- **img** – **src**, **alt**, **title**, **height**, **width** height e width regolano le dimensioni dell'immagine (se non lo specifico l'immagine viene caricata col suo formato originale, quindi se è una foto fatta con reflex avrà un formato molto grande)
 - *Elemento vuoto, non ha tag di chiusura*, tutte le informazioni sono negli attributi
 - **src**: l'indirizzo della risorsa, che può essere locale o meno
 - `` immagine presa dalla mia directory corrente (in eclipse avrò quindi un folder con le immagini salvate)
 - `` immagine presa dal web
 - **alt**: testo alternativo, da mostrare se l'immagine non è accedibile es. "a flower" se l'immagine del fiore non viene caricata
 - **title**: testo aggiuntivo mostrato quando il puntatore passa sull'immagine riquadrino come spiegazione dell'immagine
 - ``
 - **height**, **width**: dimensioni dell'immagine
 - Se nessuna delle due è indicata, si usano le dimensioni originali
 - Specificandone una l'altra viene calcolata dal browser. Entrambe: l'immagine può essere distorta
 - Valore assoluto (pixel): ``
 - Percentuale sul viewport corrente: `` 50%=occupa metà dello spazio complessivo della pagina, quindi indica la percentuale dell'img ossia del contenitore della foto
- **figure** (HTML5) contiene **img** e la descrizione relativa come **figcaption** figure e figcaption devono stare dentro un elemento di nome <figure>:
sotto l'immagine: <figcaption>This is just a flower.</figcaption>
<figure>

 <figcaption>This is just a flower.</figcaption>
</figure>

metti "." dopo img
src, quindi tipo:
<p>

</p>

iframe

- Inline frame – permette l'embedding di un'altra pagina HTML in quella corrente
- L'attributo chiave è **src**, generato dal sito ospite

il link da maps lo trovi su: condividi, incorporare una mappa, copia html (e incollalo nel tuo codice html)

```
<iframe src="https://www.openstreetmap.org/export/embed.html?bbox=9.19%2C45.46%2C9.19%2C45.46">
</iframe>
```

```
<iframe src="http://maps.google.it/maps?q=duomo+milano&output=embed">
</iframe>
```

Tabelle

- Gestione di dati in formato tabellare
- table
 - Tabella descritta come collezione di righe (dall'alto verso il basso), a loro volta descritte come collezione di celle (da sinistra a destra)
- tr
 - Riga nella tabella (table row)
- td
 - Descrive una singola cella (table datum)
 - Attributi **colspan**, **rowspan**
- th table header
 - Descrive una cella di intestazione
 - L'attributo opzionale **scope** indica se "row" o "col"

```
<table>
  <tr>
    <th></th>
    <th scope="col">Left</th>
    <th scope="col">Right</th>
  </tr>
  <tr>
    <th scope="row">Top</th>
    <td>LT</td><td>RT</td>
  </tr>
  <tr>
    <th scope="row">Bottom</th>
    <td>LB</td><td>RB</td>
  </tr>
</table>
```

	Left	Right
Top	LT	RT
Bottom	LB	RB

Rendering standard: nessun contorno a tabella e celle (CSS)

Elementi di blocco vs inline

- Blocco
 - Inizia su una nuova linea per esempio, <p> è un elemento di blocco perchè si scrive uno sotto l'altro
 - L'elemento che segue sarà su una nuova linea
 - Di solito rappresentano elementi strutturali della pagina
 - Un blocco non dovrebbe essere contenuto da un inline
- Inline
 - Contenuti in un blocco, occupano solo lo spazio necessario
 - Non implicano un andata a capo alla loro fine
 - Spesso associati a paragrafi (elemento “p”)
- Elementi generici: **div** (blocco) – **span** (inline)

differenza <p> e <div>: nel p c'è testo (che può essere formattato in un certo modo)
in un div posso mettere un altro div o uno span, ma uno span non posso metterlo in un div

id vs class

come la primary key in sql

- L'attributo **id** permette di identificare **univocamente** un qualunque elemento all'interno di una pagina
- L'attributo **class** permette di identificare un **gruppo** di elementi in un pagina
ad esempio se voglio rendere gialli un gruppo di elementi uso class per farli gialli in blocco
- L'uso di class e id è fondamentale nell'interazione tra HTML con CSS e JavaScript

Interazione con utente

utente compila form per registrarsi

- L'elemento **form** è uno tra i principali strumenti per gestire l'interazione con l'utente
 - Nelle web app classiche, hanno lo scopo di permettere l'invio di dati al backend
- window+object, pulsante
- Il form contiene **widget** (elementi HTML visualizzati in modo standard), ognuno dei quali è usato per generare un parametro con i dati da inviare

TCP/IP è uno dei protocolli di base che permettono a due macchine di interagire e scambiarsi dati in un certo formato.
HTTP è un protocollo con cui due macchine interagiscono. ora si usa HTTPS (secure) per l'importanza crescente data alla sicurezza sul web.
per mandare dati al server (tipo AS Tomcat): GET o POST. sono due comandi che ci permettono di interagire tra la nostra macchina e la macchina remota (get=cartolina con pochi dati visibili, post=busta chiusa con lettera con tanti dati privati; quindi uso HTTP e poi faccio post, non get)

sia nel get che nel post l'indirizzo è visibile, ma a differenza del get, nel post i parametri sono invisibili.

es. GETx?y=123&z=ciao : voglio mandare a un browser x un parametro y con valore 123 e un parametro z con valore ciao.

x è la action (servlet destinatario della request) a cui passo quei due parametri che saranno visibili nella barra dell'indirizzo;

quindi se devo programmare una pagina in cui l'utente deve immettere

username e password, visto che non voglio che essi siano visibili, faccio un post, così non compare nè nella barra dell'indirizzo nè nella history.

Request – Response

request viene spedita dal browser via HTTP all'Application Server (nel nostro caso Tomcat). Tomcat ha in sé del codice (servlet) che genererà la response che verrà mandata come response al browser, che poi fa il rendering della response con un'altra pagina HTML.

- Il submit di un form genera una request che viene indirizzata al server usando il protocollo HTTP specificando
 - Metodo usato, tipicamente GET o POST
 - URL destinatario
 - Parametri associati, visti come coppie name → value
- Il server gestisce la request e alla fine genera una response che viene ritornata al chiamante
- Il browser mostra il risultato all'utente

form

- Gli attributi fondamentali di un elemento **form** sono:

con l'action specifichiamo la servlet da eseguire

- **action**: URL dove devono essere mandati i dati

con method specifichiamo che tipo di comando (get, post) voglio eseguire. se non metto nulla è get

- **method**: quale metodo HTTP deve essere usato per spedire il messaggio (default GET)

nell'action c'è l'indirizzo della risorsa destinataria della request

```
<form action="/comment" method="post">
```

<div> div=scatola che contiene i widget

<label for="name">Name:</label>

riga di immissione

widget <input type="text" id="name" name="sender"> name sarà il nome del parametro che manderò alla webapp

</div>

<div>

<label for="msg">Message:</label>

<textarea id="msg" name="message"></textarea>

</div>

<div>

<button type="submit">Send</button>

</div>

</form>

Submit di un form

- In questo esempio l'input dell'utente avviene via:
 - `input-text` (stringa di testo)
 - `textarea` (blocco di testo)
- L'attributo `name` in ogni widget determina l'associazione con il parametro passato al server
- Le `label` chiariscono il ruolo del widget associato
 - L'attributo `for` collega una label al controllo con quell'`id`
- Il `button-submit` reagisce a un click dell'utente eseguendo l'azione del form

metodo

risorsa/URL
(in questo caso non aprirà nessuna URL perché è /comment che è invalido)

nel caso fosse stato get, dà errore ma nella barra dell'indirizzo c'è scritto sender e message (mentre con post appare solo comment)

```
<form action="/comment" method="post">
  <div>
    <label for="name">Name:</label>
    <input type="text" id="name" name="sender">
  </div>
  <div>
    <label for="msg">Message:</label>
    <textarea id="msg" name="message"></textarea>
  </div>
  <div>
    <button type="submit">Send</button>
  </div>
</form>
```

slide dopo

input non ha closing tag

qui dentro posso scriverci il testo

input text (et al.) – textarea

value="ciao" required maxlength="5"/>

- **input**

se non specifico il type, come default metterà text

- *Non ha closing tag*, per assegnare un valore di default si usa l'attributo **value**

L'attributo **placeholder** visualizza una indicazione per l'utente su quello che ci si aspetta come input

- Se è un parametro obbligatorio si può usare la validazione HTML5 con l'attributo **required**
- L'attributo **maxlength** fissa la lunghezza massima del valore
- L'attributo **type** determina il suo tipo specifico, tra cui:

parametro

valore di default associato al parametro user

- **text** (default) `<input type="text" name="user" value="Bob" maxlength="30" />` imposta che tipo di testo e con che limite di lunghezza ritorna al server

pwd=password inserita

- **password** (dati sensibili) `<input type="password" name="pwd" maxlength="30" required />` compaiono i pallini quando scrivo password e devo per forza scrivervi qualcosa (c'è required)

- **hidden** (parametro nascosto) `<input type="hidden" name="invisible" value="notShowed" />` si usa con java enterprise

- **date** (scelta di un giorno) `<input type="date" name="milestone" />` permette di scegliere una data da un calendarietto

- **textarea**

riquadro

- Blocco di testo su più righe, tra open e *close tag* si può inserire il testo di default

`<textarea name="comment">Enter your comment here.</textarea>`

input radio

- Scelta di una opzione da una lista
- L'attributo **checked** indica la scelta di default
- Al click del submit button, il radio button checked determina quale value viene associato al **name** e messo nella request

dà all'utente la possibilità di scegliere una sola opzione

```
<input type="radio" id="favJ" name="fav" value="Java" checked>  
<label for="favJ">Java</label>  
<input type="radio" id="favPy" name="fav" value="Python">  
<label for="favPy">Python</label>  
<input type="radio" id="favCpp" name="fav" value="Cpp">  
<label for="favCpp">C++</label>
```

attributo booleano

gruppo
determinato
da "name"

viene scritto cpp per non scrivere c++, perchè
"++" nella barra dell'indirizzo significa "spazio"
(es. Milano++Duomo= Milano Duomo)

input checkbox

- Scelta di più opzioni da una lista
- L'attributo **checked** indica le scelte di default
- Al click del submit button, se c'è almeno un checkbox checked, ogni valore viene associato al "name" (comune) e messo nella request

```
<input type="checkbox" id="langJ" name="lang" value="Java" checked>  
<label for="langJ">Java</label>  
<input type="checkbox" id="langPy" name="lang" value="Python">  
<label for="langPy">Python</label>  
<input type="checkbox" id="langCpp" name="lang" value="Cpp" checked>  
<label for="langCpp">C++</label>
```

gruppo
determinato
da "name"

select – option

- Scelta di una opzione da una lista a scomparsa
 - **select** fa da container e definisce l'attributo **name**
 - Selezione di più opzioni (via ctrl-click) aggiungendo l'attributo **multiple**
 - **option** definisce il **value** per ogni singola scelta
 - L'attributo **selected** specifica (un/il) valore di default

```
<select name="os">  
  <option value="none">-</option>  
  <option value="linux" selected>Linux</option>  
  <option value="windows">Windows</option>  
  <option value="macOs">MacOS</option>  
</select>
```


fieldset

- **fieldset**
 - Permette di raggruppare campi correlati, migliorando la leggibilità di un form
- **legend**
 - Descrive il fieldset corrente

```
<fieldset>
  <legend>User</legend>
  <label>First name: <input type="text" name="fname" /></label>
  <label>Last name: <input type="text" name="lname" /></label>
</fieldset>
```

messi in un box con righe di inserimento in box dopo first name e last name

CSS: Cascading Style Sheets

- 1996 World Wide Web Consortium (W3C), versione corrente: CSS3
- Separazione tra contenuto e presentazione in un documento HTML
- Lo stile è definito da regole
- Ogni regola è strutturata in
 - **Selettore**: a quali elementi si applica la regola
 - **Dichiarazioni**: come devono essere “stilati” gli elementi



HTML e CSS

- Si possono “stilare” elementi di un documento HTML

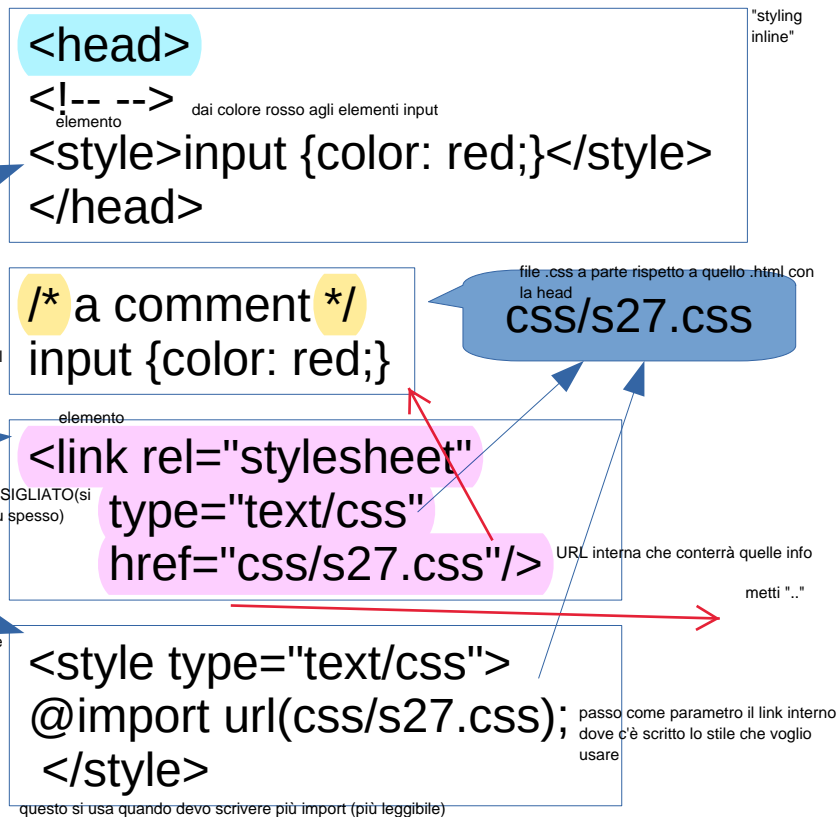
- Nella HEAD

- Definendo inline lo stile in un **elemento style** (sconsigliato in produzione)
- Definendo un collegamento a un file CSS esterno

- via un **elemento link**
- via **import** all'interno di un **elemento style**

- Nel BODY

- Nello specifico elemento usando l'**attributo style** (sconsigliato)



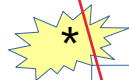
se voglio che un intero paragrafo sia in rosso scrivo:
(nell'HTML) <p class="red">
(in CSS) .red { }

se uso un id in HTML:
<p id="blue">
in CSS scriverò:
#blue { }

se ho più regole che si sovrappongono, vince l'ultima (cascading)

Selettori

- Selezione degli elementi nella pagina a cui applicare la regola:
 - Nome del tag
 - Classe, attributo class
 - Identificatore, attributo id
 - Attributo
 - Pseudo classe (hover, checked, nth-child(), ...), anchor → a:link, a:visited
 - Pseudo elemento (before, after, selection, first-letter, ...)
 - Discendenza diretta
 - Discendenza generica
 - Stesso livello, elemento successivo
- Più selettori possono essere associati a una regola
- I selettori possono essere combinati



la regola specificata nelle graffe varrà in tutti i p

```
p { ... }  
.className { ... }  
#idName { ... }  
[type=text]  
:first-child { ... }  
::before
```

voglio stilare tutti
gli input

voglio stilare la
prima voce (ad es. di una
ol)

stila solo il figlio span: <div>

```
div>span { ... }  
div span { ... }  
h1 + p { ... }
```

stila tutti gli span sia figli che nipoti:

```
<div>  
<p>  
<span>
```

stila il primo p che trovi dopo h1

```
h1, h2, h3 { ... }  
input:hover { ... }  
p.className { ... }
```

stila quando passo il mouse
su quell'input

stila tutti i p associati a quella classe (ma solo i p, non anche i div o a p associati ad altre classi)



passo col cursore su una pubblicità o su un prodotto e mi ingrandisce la pubblicità o mi mostra i dettagli del prodotto

Selettori – esempi

CSS

selettore

```
[type=text] {  
    background-color: olive;  
}  
  
[type=number] {  
    background-color: yellow;  
}  
  
input:hover {  
    color: blue;  
}
```

fai diventare background olive l'elemento che ha l'attributo type=text

diventa blu quando passo il mouse sopra al testo

html

```
<input name="firstname" type="text">  
<input name="lastname" type="text">  
<input name="age" type="number">
```

input line dedicata ai numeri

```
div span {  
    background-color: yellow;  
}  
  
div>span {  
    font-weight: bold;  
}
```

inline

p=blocco

```
<div>  
  <span>A</span> <span>B</span>  
  <p>  
    <span>C</span> <span>D</span>  
  </p>  
</div>  
<p>  
  <span>E</span> <span>F</span>  
</p>
```

Proprietà

- Alcune tra le proprietà più usate in CSS:
 - **background**: sfondo di un elemento
 - **background-color**: (yellow, #129921) ...
 - **border**: il bordo di un elemento (border: **1px solid black**;) bordo tabella
 - **border-width**, **border-color**, **border-collapse**, ... border-collapse si vede spesso nelle tabelle
 - **color**: colore del testo nell'elemento
 - **font**: proprietà del carattere per il testo nell'elemento
 - **font-size** (80%, 1.2em, 18px), **font-family** (Arial, sans-serif), **font-style** (italic), **font-weight** (bold)
 - **margin** e **padding**: spazio attorno all'elemento (esterno e interno ai bordi) es. margin: 50
 - **text-align** (center, justify): allineamento del testo al centro o al lato della casella
 - **text-transform**: (uppercase, capitalize)
 - **width**, **height**: dimensioni, quando applicabili

```
td.color {  
    text-decoration: line-through;  
} PER TESTO BARRATO
```

Esempio: tabella con CSS

```
<table>
  <tr>
    <th>Left</th>
    <th>Center</th>
    <th>Right</th>
  </tr>
  <tr>
    <td>7</td>
    <td>8</td>
    <td>9</td>
  </tr>
  <tr>
    <td>5</td>
    <td>6</td>
    <td>7</td>
  </tr>
  <tr>
    <td>1</td>
    <td>2</td>
    <td>3</td>
  </tr>
</table>
```

```
table {
  border: 2px solid black;
  border-collapse: collapse;
  Width: 50%; tabella occupa il 50% dello spazio
}

td, th {
  border: 1px solid red;
  padding: 3px;
  text-align: center;
}

th {
  background-color: lightblue;
}

td {
  background-color: lightgreen;
}
```

fa parte della triade con HTML (che permette di strutturare e scrivere pagine web) e CSS (per dare lo stile alle pagine web). il compito di javascript è di modificare la pagina corrente senza dover chiedere nulla all'application server (lavoro in locale).

js tipizzato: scrivo let x = 7, true, '7'; debolmente e dinamicamente tipizzato perchè non indico il tipo prima della variabile, ma lo si capisce da come è scritto il valore
x = 'ciao'; quando non scrivo let non sto inizializzando la variabile
js imperativo=dò ordini al browser
js funzionale=il cuore di js è la funzione
js event-driven=ad es. l'utente che fa clic sul mouse, js lega questo evento ad una funzione
come viene generato un evento, viene messo in coda agli altri (concetto di queue-first in first out):
le funzioni, in attesa che avvengano eventi simili, vengono richiamate nel momento in cui servono.
js è fatto con una struttura a coda, le funzioni scritte in js sono in attesa che si verifichi un evento.
quando un evento si verifica (es. il clic), le funzioni che servono a realizzare un output (es. fai diventare blu lo schermo dopo il clic) vengono richiamate.

JavaScript

interpreta il codice

e

- Linguaggio di programmazione interpretato, debolmente (e dinamicamente) tipizzato, multi-paradigma, imperativo, funzionale, event-driven
- Nato nel 1995 (Brendan Eich @ Netscape) per aggiungere funzionalità alla coppia HTML-CSS, è ora utilizzato anche esternamente ai browser
- Dal 1997 ECMA ne coordina lo sviluppo, con il nome ufficiale di ECMAScript
- Sostanzialmente diverso da Java

primo web browser

invece del browser può usare anche nodejs

HTML – JavaScript

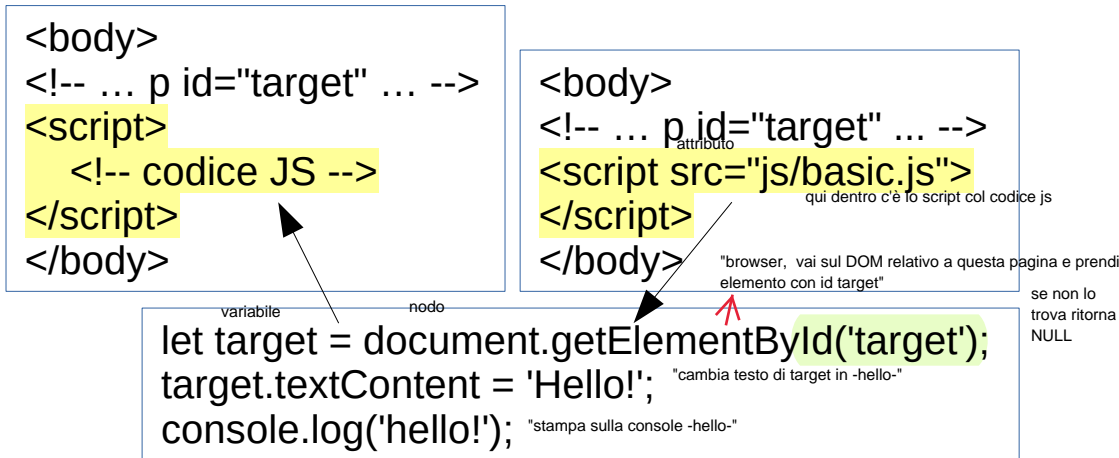
- Elemento **script**, in **head** (o ultimo elemento nel **body**)

codice js viene scritto nella head o nel body in base al progetto che si sta scrivendo

- Il codice può essere:

- Scritto direttamente nell'elemento script (sconsigliato in produzione)
- Caricato da un file JS esterno, specificato nell'attributo **src**
- Commenti JavaScript
 - `//` termina a fine riga
 - `/*` terminazione esplicita `*/`

il codice js è scritto dentro uno script .html (nel body o nell'head)



modifico il DOM (document object model, che è tutto l'albero) col codice js oppure modifico direttamente dalla console del browser (ctrl+shift+i) col codice js

Debug

- Web Developer Tools (Firefox) / DevTools (Chrome)
- Scorciatoia comune per l'attivazione: ctrl+shift+i
 - Settings (F1), Advanced settings, Disable HTTP cache
 - Tab Debugger, accesso al codice
 - Tab Console, visualizzazione log
 - Tab Inspector, HTML widget
 - Tab Style Editor, CSS

```
let x=7;  
value += x; (value=value+x)
```

Variabili

var x; (modo classico ma folle perchè ha uno scope funzionale, cioè vale su tutta la funzione e non vale come variabile locale-->hoisting) ma viene usato ancora dai programmatori
let x; (invece di mettere var bisognerebbe mettere let, ma dipende dal progetto dove si è)

- Per dichiarare una variabile si usa **let** (o var → hoisting!)
 - JavaScript è case sensitive, myname è diverso da myName

- Non si esplicita il tipo, che può essere:

mentre in java è un reference, in js string è un primitivo

- **string**: let name = 'Tim'; // apice singolo o doppi apici
- **number**: let value = 42; // sia interi sia float
- **boolean**: let flag = true; // o false
- **object**: let dog = { name : 'Zip', breed : "Alsatian" };
 - **array**: let data = [1, 'Tom', false];

se metto una stringa dentro una variabile, il browser la interpreta come un valore di tipo stringa

il browser mi ritorna undefined quando scrivo let x; quindi non la definisco
let x= null; mi ritorna null

undefined vs null

se scrivo:

if(x==3) mi ritorna undefined
perchè non l'ho creata

if(x=3) NON VA BENE perchè sto
inizializzando la x e sto facendo l'if

SENZA APICI

proprietà 1

proprietà 2

non ho la classe cane, definisco

le proprietà del cane (separandole da virgola) quando lo inizializzo

le proprietà in js sono pubbliche

un array può contenere valori di ogni tipo (int, stringa, boolean,...)

- Una variabile può cambiare il suo tipo associato nel corso della sua vita
- L'operatore **typeof** ritorna la stringa che descrive il tipo dedotto da JS (o **undefined**)
- Per dichiarare costanti si usa **const**
 - const z = 42;

simile ad instanceof di java

simile al final di java

vale la pena usarlo solo per i primitivi (come in java)

let x=3;
console.log(typeof x); (log è un metodo dell'oggetto console)
mi stampa un number

Operatori aritmetici

- **+** addizione: $2 + 3$
- **-** sottrazione: $2 - 3$
- ***** moltiplicazione: $2 * 3$
- **/** divisione: $2 / 3$ il risultato sarà frazionario
- **%** modulo o resto: $2 \% 3$ due alla terza
- ****** esponente: $2 ** 3$ // vecchio stile: `Math.pow(2, 3)` c'è l'oggetto Math per fare operazioni matematiche (in java Math è una classe)
- **++** / **--** incremento / decremento (sia prefisso sia postfisso)

Operatori di assegnamento

- Operatori che assegnano alla variabile sulla sinistra ...
 - `=` il valore sulla destra
 - `+=` la somma dei valori a sinistra e destra
 - `-=` la differenza tra il valore di sinistra e quello di destra
 - `*=` il prodotto del valore di sinistra per quello di destra
 - `/=` la divisione del valore di sinistra per quello di destra

Operatori relazionali

- Operatori che ritornano un booleano

=== stretta uguaglianza (stesso tipo e valore)

```
let a=7;  
let b='7';  
if(a==b) { non potevo mettere === perchè a e b non sono dello stesso tipo
```

!== di stretta disuguaglianza (diverso tipo o valore)

< valore sulla sinistra è minore del valore sulla destra

<= minore o uguale

> il valore sulla sinistra è maggiore del valore sulla destra

>= maggiore o uguale

!! conversione a booleano, equivalente alla funzione **Boolean()**

si preferisce usare la funzione Boolean():
invece di scrivere x!==a; (a=7 quindi true perchè altri diverso da zero-
false-è true)
scrivo y=Boolean(a);

- Gli operatori non-strict **==** e **!=** possono causare conversioni implicite

Stringa

- Una stringa è una sequenza **immutabile** di caratteri delimitata da apici singoli o doppi
- Per **concatenare stringhe** si usa il metodo **concat()** o l'operatore **+**
 - Conversione implicita da numero a stringa come in java
`'Solution' + 42 === 'Solution42'`
- Conversione esplicita da numero a stringa via **toString()**
`a.toString() === '42' // se a === 42`
- Conversione esplicita da stringa a numero via **Number()**
`Number('42') === 42`

Lavorare con stringhe

stringa è un primitivo che ha dei metodi (i primitivi sono trattati come oggetti)

- Lunghezza: `s.length`
- Accesso ai caratteri: `s[i]` // `i` in `[0, s.length-1]`
in che posizione è una sottostringa
ritorna undefined se il valore è fuori dall'intervallo
- Ricerca di sottostringa: `s.indexOf(sub)` // -1 not found
- Estrazione di sottostringa:
begin
 - `s.substring(beg, end)` // swap if `beg > end`
voglio una fetta
 - `s.slice(beg, end)` // end negativo == `len - end`
non modifico la stringa corrente, ma ne creo una nuova
- Minuscolo: `s.toLowerCase()`
- Maiuscolo: `s.toUpperCase()`
- Modifica: `s.replace(sub, other)`
- Estrazione di componenti: `s.split(',')` // da stringa ad array

Array

- Collezione di oggetti di qualunque tipo
- Numero di elementi nella proprietà `length`
- Accesso agli elementi in lettura e scrittura `data[i]`

- Scansione di tutto l'array via for loop

voglio convertire un array in stringa-->faccio il join degli elementi

- Da array a string via `join()`, `toString()`

è più simile ad una collezione che ad un array di java

- Per aggiungere un elemento: `push()`, `unshift()`
- Per eliminare un elemento: `pop()`, `shift()`, `splice()`

push per mettere un elemento nello stack

col pop tolgo
un elemento
dall'array
(che sta nello
stack)

elimina
il primo
elemento
dell'array

spacca l'array, toglie degli
elementi e sostituisce con
degli altri e ricongiunge
l'array

```
let data = [1, 'hello', [true, 42.24]];
console.log(data.length);
```

```
console.log(data[1], data[2][1]);
data[2] = false;
```

```
for(let i = 0; i < data.length; i++) {
  console.log(data[i]);
}
```

```
console.log(data.join(), data.toString());
```

```
data.pop();
data.shift();
data.push('push');
data.unshift('unshift');
```

```
if(x) {  
  ...fai qualcosa  
} else {  
  ..fai qualcosa  
}
```

Condizioni

- Molto simile a Java
 - **if – else** (if)
 - AND con **&&**, OR con **||**, NOT con **!**
i vari casi dello switch e il default finale
 - **switch – case – default**
 - Operatore ternario **?:**
- Ma ...
 - Preferito l'uso degli operatori *strict* **===** e **!==**
 - **conversione implicita a boolean** che ritorna **true** per valori che **non** sono **false, undefined, null, 0, NaN, ""** (la stringa vuota)
è false anche l'undefined, il null, lo zero, il Not a Number (es. la radice di un numero negativo o 0/0 ritornano NaN) e la stringa vuota

Loop

- Come in Java

si fa prima di eseguire il for

se è true continuo a loopare

incremento/decremento

- `for`(inizializzazione; condizione; espressione) {
 }
– `while`(condizione) {
 }
– `do` {
 }
 `while`(condizione);
– `break`;
– `continue`;

Funzione

- Blocco di codice a cui è associato un nome, definite indicando
 - la keyword **function**
 - il nome (opzionale: funzioni anonime, notazione classica e “freccia”)
 - una lista di parametri tra parentesi tonde
 - l'oggetto arguments, default per parametri **x = 0**, parametro 'rest' **...va**
 - una **lista di statement tra parentesi graffe**
- In JavaScript **sono oggetti**, e dunque possono
 - essere assegnate a variabili, proprietà di oggetti, elementi di array
 - essere passate ad altre funzioni
 - contenere altre funzioni (metodi)
- Si invoca una funzione specificando
 - il suo nome
 - i valori da associare ai parametri – se non specificati, default o undefined

first class citizen=trattate come tutti gli altri oggetti

un oggetto può avere proprietà, per cui una funzione può avere un'altra funzione

funzione senza parametri=js ritorna un undefined

let a=() {console.log('ciao');
a();} è un oggetto-funzione

```
sort(a,b)
if(a>b){
  return -1;
}else if(a<b){
  return +1;
}else{
  return 0;
}
```

```
function f() {
  console.log('hello');
}
```

non ritorna nulla, quindi mette undefined

```
function g(a, b) {
  return a + b;
}
```

non sappiamo se sia una stringa o un numero
(dipende da cosa mi ha passato l'utente)

```
let f1 = function(a, b) {
  return a + b;
}
```

freccia grassa
questi parametri sono associati ad a+b

```
let f2 = (a, b) => a + b;
```

```
f();
```

```
let result = g(3, 5);
```

invoco funzione g passandogli
parametri 3 e 5

AJAX e XMLHttpRequest

- **A**synchronous **J**avaScript **A**nd **X**ML
- Uso dell'oggetto XMLHttpRequest per comunicare con il server (XML, JSON, testo semplice, ...) senza lasciare la pagina corrente
- Dopo aver creato un oggetto XMLHttpRequest
 - Si definisce una callback in onload (o onreadystatechange)
 - Si invoca open() per definire la risorsa richiesta sul server
 - E infine send()

jQuery

insieme di funzionalità messe a disposizione del programma, così non devo scriverlo ogni volta ma ce l'ho già lì (è un file .js)

- **Libreria JavaScript** progettata per semplificare la gestione del DOM (Document Object Model) di pagine HTML
- Creata da John Resig nel 2006
- Download da <https://jquery.com/download/> alternativa 1

```
<script src="js/jquery-3.4.1.min.js"></script>
```

min=minimizzato per ridurre lo spazio

nello script prima scrivo la libreria e poi il codice js da eseguire

content delivery network

- CDN <https://code.jquery.com/>

```
<script src="http://code.jquery.com/jquery-3.4.1.min.js"></script>
```

alternativa 2: scarico jquery "da remoto" accedendo alla macchina più vicina (es. sydney da melbourne invece che da NY)

- Documentazione <https://api.jquery.com/>

L'evento ready

si può scrivere indifferentemente jQuery o \$

```
jQuery(document).ready(function() {  
    // ...  
});
```

```
$(document).ready(function() {  
    // ...  
});
```

```
$(function() {  
    // ...  
});
```

dentro la funzione anonima metto il codice che voglio che acceda al DOM

- Prima di eseguire uno script, bisogna assicurarsi che il documento sia pronto
- Il metodo ready() di jQuery ha come parametro una funzione in cui possiamo mettere il nostro codice
- Il dollaro è l'alias comunemente usato per la funzione jQuery()
- Forma abbreviata equivalente

Selezione di elementi

- Wrap jQuery di elementi via selettore CSS

si possono usare tutti i selettori usati in CSS

tag: \$('textarea')

id: \$('#myId')

classe: \$('.myClass')

lista di selettori: \$('div,span')

...

- Numero di elementi selezionati: length
 - Esempio: numero di div nella pagina: \$('div').length

Creazione di elementi

- Passando il relativo codice HTML si può creare un elemento, arricchirlo e inserirlo nel documento
- Esempio:
 - Crea un div contenente 'Hello'
 - Stilalo assegnando un colore al suo testo
 - Appendi l'elemento al body della pagina

frammento di codice HTML con un div con scritto hello su cui chiamo il metodo css di jquery per applicare lo stile rosso (cioè rendere il testo hello rosso), poi metti questa riga alla fine del body

```
$('<div>Hello</div>').css({color: 'red'}).appendTo('body');
```

click e dblclick su un bottone

- Risposta a evento click e double click

// override del comportamento dei link in una pagina

```
vai a prendere sulla pagina tutti gli elementi a (tutti quelli che contengono anchor=link) $('a').click(function(event) { quando qualcuno clicca su questo link chiama un alert. questa funzione è in attesa finché l'evento non accade  
    alert("You should not use any link on this page!");  
    event.preventDefault(); quando uno clicca non succede nulla (non va alla pagina)  
});
```

// double-click detector

```
$('html').dblclick(function(e) {  
    console.log('Double-click detected at ' + e.pageX + ', ' + e.pageY + '\n');  
});
```

posso aggiungere o togliere classi via jquery

un elemento può avere più classi:

es. `<div id='x' class='a b c'>` una classe gestisce il bordo, una il background color ecc

L'attributo class

le classi si trovano nel file .css

```
(es. .red {  
  color: red;  
})
```

in .js trovo invece:

```
$(function() {  
  $('#colorOn').click(function() {  
    $('#msg1').addClass('red');  
  });  
});
```

js usa gli stili (es. colore) che css fornisce

aggiungi classe red a elemento #msg1

- `addClass()`

```
$('#msg1').addClass('red');
```

- `removeClass()`

```
$('#msg1').removeClass('red');
```

scambio

- `toggleClass()`

```
$('#msg2').toggleClass('red');
```

se c'era la scambia, se non c'è la mette

- `hasClass()`

```
$('#msg3').hasClass('red');
```

ritorna un booleano

Getter e setter

- `html()` – Mantiene la formattazione HTML

- `text()` – Testo puro

setter jQuery, prendi l'elemento con questo id e metti come testo il seguente:

```
$('#signature').text('Hello by JQuery');
```

- `val()` – Accesso al valore in input

```
$('#msg').val('Something');
```

- `css()`

per settare o leggere lo stile di un elemento

getter-ritorna un font size

```
let cur = parseInt($('#msg').css('font-size'));
```

2

leggi la font size dell'elemento msg, ho fatto la parse int e l'ho memorizzato come stringa nella variabile cur.
cur avrà come font size la stessa di msg

```
$('#msg').css('font-size', cur * 2);
```

setter-perchè ha due parametri (font size e quanto lo voglio raddoppiare)

Bootstrap

- Framework CSS per lo sviluppo web front-end
(più modulo JavaScript opzionale)
- Progetto interno di Twitter, 2011
- Download da <https://getbootstrap.com/>
`<link rel="stylesheet" href="css/bootstrap.min.css">`
- CDN da <https://www.bootstrapcdn.com/>
`<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">`

Setup

- Assicurarsi che il browser interpreti la pagina come HTML5
 - `<!doctype html>`
- Head
 - Inserire i seguenti **meta**
 - `<meta charset="utf-8">`
 - `<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">`
 - Inserire il **link** a Bootstrap

Container

- Due tipi di container
 - **container**, lunghezza fissa per ogni breakpoint
 - **container-fluid**, è sempre il 100% del viewport

```
<div class="container">  
  <h1>Hello from Bootstrap</h1>  
</div>
```

```
<div class="container-fluid">  
  <h1>Hello from Bootstrap</h1>  
</div>
```

Grid

- All'interno di un container, gli elementi sono organizzati in righe e colonne
- Un div di classe **row** per ogni riga
- Un div di classe **col** per ogni cella, implicitamente tutte della stessa dimensione

```
<div class="container-fluid">  
  <div class="row">  
    <div class="col">1/1</div>  
    <div class="col">2/1</div>  
    <div class="col">3/1</div>  
  </div>  
  <div class="row">  
    <div class="col">1/2</div>  
    <div class="col">2/2</div>  
    <div class="col">3/2</div>  
  </div>  
</div>
```


Breakpoint

- La dimensione del viewport viene categorizzata in **breakpoint**
extralarge (**xl**), large (**lg**), medium (**md**), small (**sm**)
- Ogni col può avere una dimensione in dodicesimi

```
<div class="row">  
  <div class="col-sm-2 col-md-3 col-lg-5 col-xl-1">1</div>  
  <div class="col-sm-4 col-md-3 col-lg-1 col-xl-5">2</div>  
  <div class="col-sm-4 col-md-3 col-lg-1 col-xl-5">3</div>  
  <div class="col-sm-2 col-md-3 col-lg-5 col-xl-1">4</div>  
</div>
```

Table

- Per stilare un **elemento table** lo si mette in un **container**, gli si applica la **classe table** e ...
 - table-borderless
 - table-dark
 - table-striped
 - table-bordered
 - table-hover
 - table-sm
- Classi per thead
 - thead-dark, thead-light
- Classi per table, th, tr, td
 - table-success, table-danger, table-info, table-warning, ...

```
<table>
  <thead>
    <tr>
      <th scope="row">\</th>
      <th scope="col">Left</th>
      <th scope="col">Right</th>
    </tr>
  </thead>
  <tr>
    <th scope="row">Top</th>
    <td>X</td>
    <td>Y</td>
  </tr>
  <tr>
    <th scope="row">Low</th>
    <td>1</td>
    <td>2</td>
  </tr>
</table>
```