Web

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

Tomcat: applicativo su cui andremo a mettere la nostra web application (permette di scrivere codice java per rispondere alle richieste dell'utente), che verificherà anche se abbiamo il diritto di accedere a quella pagina. Riceve le richieste e dà le risposte. SWAT è l'app che corre dentro Tomcat

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

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
  - Target runtime: Server view (Window → Show View → Servers)

HTML: HyperText Markup Language

- Tim Berners-Lee @CERN ~1990
- World Wide Web Consortium (W3C) HTML5 2014
- Descrive come rappresentare pagine web
- Il rendering è responsabilità del browser
  - Chrome
  - Firefox
  - Safari
  - …
- Struttura ad albero, ogni nodo è un elemento
  - DOM: Document Object Model

inventato al CERN, per scambiarsi informazioni tra scienziati che stavano studiando sul bosone di x.
ci permette di creare documenti collegati tra di loro come ipertesti: idea chiave, i link.
linguaggio marcato, differente da un linguaggio turing completo imperativo, in quanto descrittivo: descrive una pagina e dice al browser di fargliela vedere (rendering). ogni browser lo implementa a modo suo. Molto simile ai file xml (come il pom), con un tag di apertura e uno di chiusura, che contiene una struttura ad albero. **HTML** in questo caso è il nodo principale, dentro abbiamo i due nodi **head** e **body**. A loro volta ci sono due nodi **meta** e **title**, ma meta non ha chiusura quindi si deduce che finisca lì, proprio come **"p"** sotto body.

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

hello.html

dopo aver ricevuto la richiesta di render, il dom me la rende struttura ad albero come in figura. Viene generato quando il browser legge la pagina.

Elemento

delimitato dai tag (e infatti spesso chiamato tag per abbreviare), altro modo per chiamare il nodo in html. gli elementi meta non hanno tag di chiusura RICORDALO. l'elemento contiene un nome, ma possono esserci altri elementi, come *charset* nel caso di meta è un attributo,modifica in qualche modo l'elemento (nome attributo, =, " "). Dentro gli elementi ci sta testo o altri elementi.

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: <!-- … -->

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

due parti del documento, se io non li mettessi il browser li metterebbe automaticamente. sta tutto dentro html.

head vs body

- html
  - Contiene l'intero codice HTML della pagina
- head
  - Informazioni *sulla* pagina
- body
  - Informazioni *nella* pagina
  - Contenuto che vogliamo mostrare all'utente

informazioni sul documento, che interessano al browser più che all'utente. ad esempio, se mi dimentico di specificare utf-8 non riesco a codificare i caratteri perché il browser mette dentro una tabella diversa.

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

conversione numero-carattere (0,1)

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 charset="utf-8">

    descrizione della pagina

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

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

    chi si occupa di questi meta è un SEO, cioè si occupa dei ranking delle pagine web. per approfondire, ultimo link.
    i SEM studiano dove mettere le pubblicità a seconda delle parole chiave inserite
  - Per altri tipi di metadati vedi
    - The Open Graph protocol https://ogp.me/

Testo

deriva tutto dalla comunicazione scientifica tra gli scienziati.
h1 sta per head 1, titolo, e così via sempre più piccolo fino al 6 in base alla grandezza che voglio.
il paragrafo viene delimitato anche lui dai tag: sa da solo che ogni parola è separata da uno spazio, quindi se ne inserisco di più o uso ad esempio tag è come se non avessi fatto niente, il browser decide da solo. Se voglio forzare l'andata a capo basta usare < br >, break oppure <hr> cioè proprio una riga orizzontale (si può trovare anche la dicitura col tag di chiusura hr/ per via di una versione futuristica che è stata poi abbandonata).

- h1..h6
  - Titoli (heading) di parti del testo, di solito ci si aspetta un solo H1 per pagina
- p
  - Paragrafo, unità di base per la suddivisione del testo
- b, i, u, sup, sub, …
  - Formattazione del testo, (bold → grassetto), (italic → corsivo), 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/>

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 scorciatoia!!

Liste

- ol
  - Lista ordinata in cui ogni voce ha un indice crescente
  - L'elemento ol contiene un elemento li (list item) per ogni voce
- ul
  - Lista senza ordine, come ol ogni voce è un li, ma pallino (o altro) invece di indice
- dl
  - 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

 inizia la lista ordinata, e ogni elemento sarà preceduto da <li>.

Link

modo per collegare una parte della mia pagina con un'altra risorsa nel web o locale.
#top vuol dire l'elemento che ha ID=top.

l'attributo title è il pop up piccolo che appare quando col cursore passo sopra al link.

Gestione dell'ipertestualità nelle pagine HTML

- a – href
  - anchor to an hypertext reference, "ancora" l'elemento ad una risorsa definita nel suo attributo href
    - risorsa interna: index page
      vado a finire qui
      se clicco qui
    - elemento nella pagina corrente
      - 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 il contenitore dell'immagine, lo spazio vuoto in html in cui voglio visualizzare l'immagine. non è quindi l'immagine stessa: per vederla la metto nel source (src) e scrivo dov'è contenuta, cioè in locale oppure sul web. heigth e width sono fondamentali per la dimensione dell'immagine, della quale, se non specifico unità di misura intendo pixel (viewport: l'area di visualizzazione, quindi il 50% vorrà dire che la mia foto prenda metà pagina).

- img – src, alt, title, height, width
  - *Elemento vuoto, non ha tag di chiusura*, tutte le informazioni sono negli attributi
  - src: l'indirizzo della risorsa, che può essere locale o meno
    - <img src="pic/flower.jpg">
    - <img src="https://developer.mozilla.org/static/img/opengraph-logo.72382e605ce3.png”>
  - alt: testo alternativo, da mostrare se l'immagine non è accedibile
  - title: testo aggiuntivo mostrato quando il puntatore passa sull'immagine
    - <img src="pic/flower.jpg" alt="A flower" title=”A flower”>
  - 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): <img src="pic/flower.jpg" width="150" height="150">
    - Percentuale sul viewport corrente: <img src="pic/flower.jpg" width="50%">
- figure (HTML5) contiene img e la descrizione relativa come figcaption

iframe

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

```
<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&amp;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
  - 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>
```

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

	Left	**Right**
Top	LT	RT
Bottom	LB	RB

Elementi di blocco vs inline

- Blocco
  - Inizia su una nuova linea
  - 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)

Blocco: cerca di occupare tutto lo spazio sia in larghezza che in altezza, l'elemento inizia su una nuova linea andando a capo. un blocco può contenere altri blocchi.(es. <p>)

Inline: gli elementi sono consecutivi, minimizzando lo spazio occupato.(es. <b>, che indica il grassetto nel <p>)

div e span non hanno connotazioni di nessun tipo, sono generici (invece p e b indicano cose ben precise): lo span sta dentro al div ma non viceversa.

http://localhost:8080/mswat/s25.html è il formato in cui devo leggere le mie pagine

sono attributi. id identifica un solo elemento; class invece mi identifica un gruppo di elementi. scopo: dire che questi elementi diversi hanno qualcosa in comune

id vs class

- 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
- L’uso di class e id è fondamentale nell’interazione tra HTML con CSS e JavaScript

Interazione con utente

- 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
- Il form contiene widget (elementi HTML visualizzati in modo standard), ognuno dei quali è usato per generare un parametro con i dati da inviare

una finestra in cui l'utente può mettere dentro dei dati. ricevo le informazioni dell'utente , il browser genera la request, poi invio all'application server che le elabora: rispondo sempre al browser, che fa il rendering della response creando una nuova pagina html.
Nell'action (contenuta nel form) specifico l'indirizzo.
vedi dopo--->

Request – Response

internet: computer collegati tra di loro tramite protocollo tcp/ip, che si scambiano bit. l'http è un protocollo per il trasferimento di dati tra due macchine, di livello più alto. https: secure, è tutto criptato, sta diventando lo standard.

- 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

se voglio sapere x avendo y come parametro, l'indirizzo generato dal browser è : get x? y=...(metto i parametri). x quindi l'avrò specificata in <form action "x"). col post vedrei solo post x , i dati sarebbero criptati nel payload.

comandi http che ci permettono di interagire tra la nostra macchina e la macchina remota. nella get mettiamo sia l'indirizzo che i dati che vogliamo mandare. Il post è come una lettera col sigillo, non si può aprire: i dati sono criptati, ma l'indirizzo c'è in tutte e due. Se lavoro in https quindi è inutile fare la get perché i dati sono visibili e non rispettano la Security. In più, conviene usare la get quando hai pochi valori

- 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:
  - action: URL dove devono essere mandati i dati
  - method: quale metodo HTTP deve essere usato per spedire il messaggio (default GET)

la risorsa a cui voglio accedere. se non metto il metodo fa automaticamente una get

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

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

```
<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>
```

etichetta

gli diamo un name perché sarà il nome del parametro che darò alla mia web app Sender è quello che inserisce l'utente come nome. input: l'area in cui possiamo mettere dentro il sender

il widget textarea ha un tag di chiusura, input no

chiamando il submit parte l'esecuzione del form. quindi per la uri "comment" parte un post.

input text (et al.) – textarea

- input se non specifichiamo il type, si sottointende che sia un text
  - *Non ha closing tag*, per assegnare un valore di default si usa l'attributo value

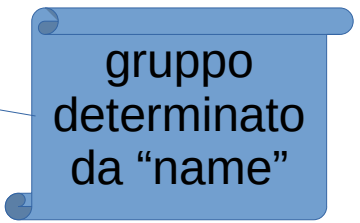
    L'attributo placeholder visualizza una indicazione per l'utente su quello che ci si aspetta come input
  - Se è un parametro obbligatorio si può usare la validazione HTML5 con l'attributo required
  - L'attributo maxlenght fissa la lunghezza massima del valore
  - L'attributo type determina il suo tipo specifico, tra cui:
    - text (default) <input type="text" name="user" value="Bob" maxlength="30" /> quando facciamo il submit, name sarà il nome del parametro generato. value è il valore di default che metto nella casella di testo
    - password (dati sensibili) <input type="password" name="pwd" maxlength="30" required />
    - hidden (parametro nascosto) <input type="hidden" name="invisible" value="notShowed" />
    - date (scelta di un giorno) <input type="date" name="milestone" /> una specie di calendario che ci aiuta a selezionare la data
- textarea diverso dall'input perché è un riquadro multiriga dove posso inserire un testo e perché ha closing tag. non ha value, quindi per mettere il testo lo devo esplicitare tra le angolari
  - 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 clicchi uno e si deleziona l'altro, devono avere tutti lo stesso nome
- 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

```
<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>
```

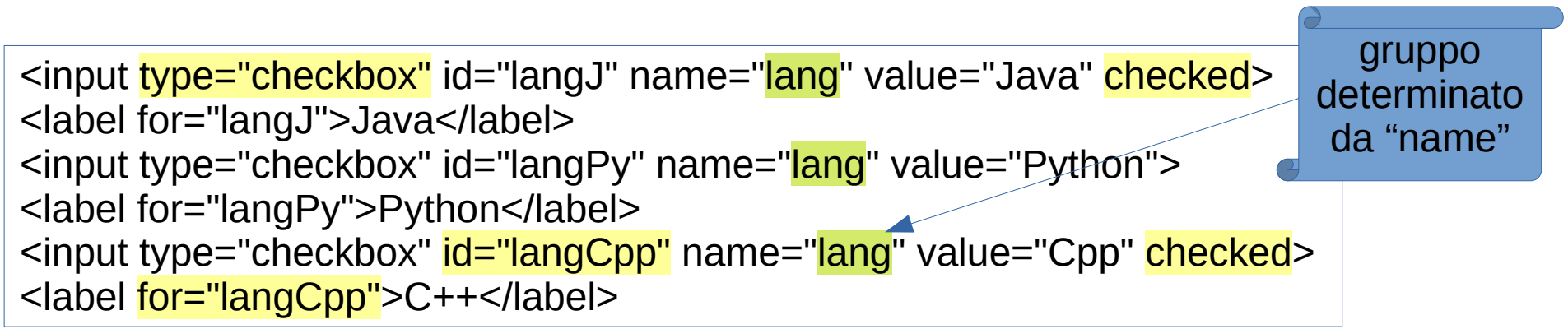


input checkbox

con questa l'utente può anche deselezionare la sua scelta se cambia idea, con la radio no.

- 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>
```



più utile della radio se voglio avere tante opzioni, sarebbe noioso metterle tutte in fila

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

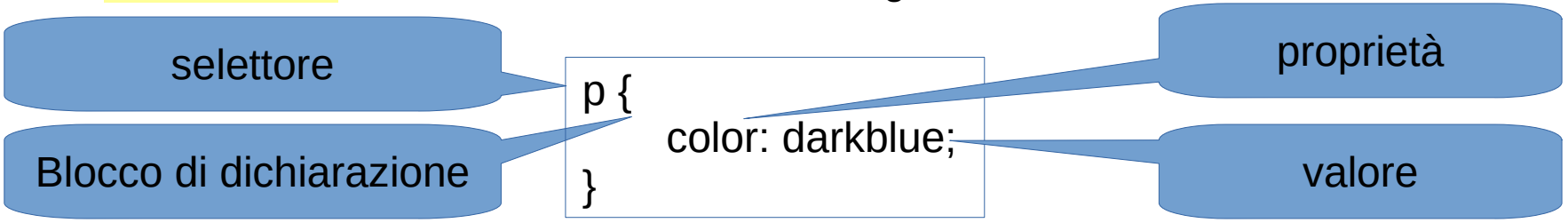
```
<fieldset>
    <legend>User</legend>
    <label>First name: <input type="text" name="fname" /></label>
    <label>Last name: <input type="text" name="lname" /></label>
</fieldset>
```

cascading perché se ho due proprietà che si sovrascrivono vince l'ultimo

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

lo styling all'interno di html diventava oneroso, quindi sempre dal cern è nato il css che include tutto quello che riguarda lo stile e la presentazione. creiamo delle regole in cui definiamo come venga stilato un elemento o un gruppo di elementi. ogni dichiarazione è in formato **prorpietà:valore**



IP CONFIG su riga di comando ti dice l'indirizzo della tua macchina.
scrivendo ping e poi l'indirizzo accedo all'altro pc dal quale voglio vedere cose

sarebbe bene avere un file html e uno css. se devo modificare qualcosa di piccolo posso fare come nel primo riquadro: essendo una regola la mettiamo nella head.

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)

```
<head>
<!-- --> agli elementi input metti il colore rosso come impostazione testo
<style>input {color: red;}</style>
</head>
```

```
/* a comment */
input {color: red;}
```

css/s27.css

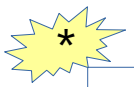
```
<link rel="stylesheet"
      type="text/css" documento di testo che descrive il css
      href="css/s27.css"/>
```

```
<style type="text/css">
@import url(css/s27.css);
 </style>
```

gli passo come parametro il nome del file a cui voglio accedere. è comodo se vogliamo importare più css

Selettori

- Selezione degli elementi nella pagina a cui applicare la regola:
  - Nome del tag tutti gli elementi di un certo tipo
  - Classe, attributo class ad esempio creo la classe "red" in html dove il testo è in rosso. in css scrivo .red e richiamo quella classe
  - Identificatore, attributo id funziona come la classe, ma col # invece che col "." e si riferisce a un solo elemento
  - Attributo solo gli elementi con un certo attributo e un certo valore (type attributo e text valore ad esempio)
  - 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




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

ad esempio il primo elemento di una lista

lo span che è direttamente dentro a un div, altrimenti no

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

tutti gli span che sono dentro ad un div

un paragrafo subito dopo l'h1 e stop

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

quando passo col mouse sopra un input, voglio che succeda...

a tutti i p che hanno questa classe, non a tutti i p

Selettori – esempi

parentesi quadra: vammi a vedere gli elementi che hanno come attributo text

```
[type=text] {
    background-color: olive;
}
```

il testo all'interno sono solo cifre

```
[type=number] {
    background-color: yellow;
}

input:hover {
    color: blue;
}
```

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

```
div span {
    background-color: yellow;
}

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

```
<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;)
    - border-width, border-color, border-collapse, ... se voglio che due bordi diventino uno
  - 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)
  - text-align (center, justify): allineamento del testo
  - text-transform: (uppercase, capitalize)
  - width, height: dimensioni, quando applicabili

Esempio: tabella con CSS

```html
<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>
```

```css
table {
    border: 2px solid black;
    border-collapse: collapse;
    Width: 50%;
}

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

th {
    background-color: lightblue;
}

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

JavaScript

- Linguaggio di programmazione interpretato, debolmente (o 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

HTML – JavaScript

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

- 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 */

```
<body>
<!-- … p id="target" … -->
<script>
   <!-- codice JS -->
</script>
</body>
```

```
<body>
<!-- … p id="target" ... -->
<script src="js/basic.js">
</script>
</body>
```

```
let target = document.getElementById('target');
target.textContent = 'Hello!';
console.log('hello!');
```

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

Variabili

- 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:
  - 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];

primitivi

undefined vs null

- 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 constanti si usa const
  - const z = 42;

Operatori aritmetici

- + addizione: 2 + 3
- - sottrazione: 2 - 3
- * moltiplicazione: 2 * 3
- / divisione: 2 / 3
- % modulo o resto: 2 % 3
- ** esponente: 2 ** 3 // vecchio stile: Math.pow(2, 3)
- ++ / -- 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)
  - `!==` 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()`
- 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

    '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

- Lunghezza: s.length
- Accesso ai caratteri: s[i] // i in [0, s.length-1]
- Ricerca di sottostringa: s.indexOf(sub) // -1 not found
- Estrazione di sottostringa:
  - s.substring(beg, end) // swap if beg > end
  - s.slice(beg, end) // end negativo == len - end
- 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

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

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

```
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');
```

Condizioni

- Molto simile a Java
  - if – else (if)
  - AND con &&, OR con ||, NOT con !
  - 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)

Loop

- Come in Java
  - 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

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

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

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

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

```
f();

let result = g(3, 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

- 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/

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

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

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

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

L’evento ready

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

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

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

- 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

  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

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

click e dblclick

- Risposta a evento click e double click

```
// override del comportamento dei link in una pagina
$('a').click(function(event) {
    alert("You should not use any link on this page!");
    event.preventDefault();
});
```

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

L’attributo class

- addClass()

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

- removeClass()

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

- toggleClass()

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

- hasClass()

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

Getter e setter

- html() – Mantiene la formattazione HTML
- text() – Testo puro

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

- val() – Accesso al valore in input

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

- css()

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

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 Boostrap

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>
```