

Html è un linguaggio markup che serve per definire delle pagine che tramite il browser le visualizza nel web e relazionarle come ipertesti e quindi collegati per mezzo di link.

Come fa il browser a rappresentare le pagine web?! Trasforma il mio documento di testo in DOM che è un albero dove ogni nodo rappresenta un elemento, con la radice che è l'elemento html. sotto html ci sono due nodi head e body e sotto di loro altri nodi

In head metto tutte le info sulla pagina. es: css, title, script (se usiamo jquery), anche i meta (per definire i caratteri utilizzati (es ts8), linguaggio (ita o ingl..), keyword (per farsi che il motore di ricerca) Tutto ciò che rende la pagina più accessibile, più gradevole

nel body: tutto ciò che c'è dentro la pagina.
Lo script va nel body come ultimo elemento

Cos'è un elemento?
Come TESTO:
E' tutto ciò che sta tra il tag di apertura ed il tag di chiusura e si dirà elemento di tipo x se dentro il tag c'è x.
Ci sono delle eccezioni e quindi non hanno il tag di chiusura
IN MEMORIA
Elemento per il browser è un nodo dell'albero

dentro il tag (<-->) ci dice il tipo di elemento che andremo ad aprire.
dentro troveremo sia testo oppure altri elementi

Lista: ordinata o no, e come lo scrivo

Cos'è un link e come funziona

Com'è organizzata la tabella

Cos'è un div: lo usiamo per rappresentare un blocco di elementi.
occupa tutto il viewport
Cos'è pan: è un inline nel senso che non mi occupa tutta la pagina, ma solo lo spazio necessario per l'elemento

id e class SONO ATTRIBUTI mi definiscono meglio il tag.
ID è un identificatore univoco dell'elemento. per pagina id unici, mai ripetuti
CLASSE è un identificatore di gruppo (di + elementi)

SELETTORE è tutto ciò che collega l'elemento al css?!

Per far interagire l'utente con la pagina: FORM. ci sono i widget che sono specificati dall'attributo "name" che è quello che identifica l'elemento per il server.
Invio tutto ciò che ha inserito l'utente al server tramite il button.
Action è l'url dove devono essere mandati i dati
Method= dice che tipo di metodo HTTP bisogna usare per spedire il testo. Noi usiamo GET o POST.
GET: si vedono i parametri inviati nell'url.
POST è il modo di inviare i parametri criptati

Select - option da spiegare

CSS è un linguaggio che ci permette di definire lo stile della pagina html.
Al suo interno ci sono regole che vanno messe dentro un blocco di selezione = {} e le metto dopo aver specificato il selettore. dentro il blocco ci saranno delle regole= "proprietà:valore;"
INTERAZIONE tra HTML e CSS va nella head

BOOTSTRAP è una libreria CSS quindi ci mette a disposizione gli stili per diversi tipi di browser. metto un link nella head.
Ha il container fluid una classe che fa sì che la pagina si adatti al view port

JavaScript: è un linguaggio di programmazione nato per lavorare con html per rendere le pagine attive. (più dinamiche). si possono modificare le pagine in diretta.
Mi gestisco una coda di eventi, ogni volta che succede qualcosa lo metto nella coda, se ho una funzione che dipende da un altro evento (es click del bottone), cioè che è interessato ad un evento, viene avvisato quando succede l'evento (es qualcuno ha cliccato sul bottone) e da lì parte una funzione.
es clicca sul bottone ed appare la tabella

AJAX è asincrono e quindi non è come il form che deve aspettare che si concluda la funzione precedente per fare la successiva.
Inoltre rimane su quella pagina, non esce da quella.

JQUERY è una libreria javascript, quindi un file di testo. per usare jquery nella nostra pagina lo metto nella head tramite <script> </script>.
Ci dovrebbe semplificare la vita nell'uso del DOM

Funzione READY di JQuery. è un evento che fa sì che il codice venga eseguito solo quando la pagina è stata completamente caricata dal browser, cioè siamo sicuri che tutto il dom è disponibile.!

Link utili

- Gli standard per il web del W3C

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

- MDN Mozilla Developer Network

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

Sviluppo su Eclipse

- Plugin Web Tools Platform
 - Eclipse Web Developer Tools
 - HTML Editor (WTP)

Quando l'utente chiede è necessario un Architectick server (noi usiamo Jetty)

Devo essere sicura che eclipse abbia dentro Jetty, su market place

- File, New, Static Web Project

Nella cartella su eclipse, ci sono immagini in pic, nel web content (generato da eclipse) vediamo le pagine html

- Target runtime: HTTP Preview (usa il server Jetty di Eclipse)
 - Server view (Window → Show View → Servers)
 - WebContent: dove vanno messi i file sorgenti
- Progetto Eclipse di esempio su GitHub
 - <https://github.com/egalli64/eswp>

HTML: HyperText Markup Language

- Tim Berners-Lee e @CERN ~1990
 - Utilizzato per condividere elevate quantità di dati, collegati con altri dati. es mettendo il link
- 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

Commento: inizia !-- e finisce con --

```
<!DOCTYPE html>
<!-- my hello page -->
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

hello.html

Elemento

Singolo componente di un documento HTML

- Normalmente delimitato da **open** – **close** tag
- Può contenere testo e altri elementi
- Può avere attributi nella forma nome="valore"
- “!” 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>
    <title>Hello</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Nodo 1: head, mi da informazioni sulla pagina (carattere, lingua, titolo)

Il titolo non si vede nella pagina ma nel nome della scheda

Nodo 2: le informazioni che voglio dare a chi mi accede a questa pagina

head vs body

- html
 - Contiene l'intero codice HTML della pagina
- head
 - Informazioni *sulla* pagina
- body
 - Informazioni *nella* pagina

```
<!DOCTYPE html>
<!-- my hello page -->
<html>
  <head>
    <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, si consiglia l'uso di charset, che rappresenta l'encoding usato nella pagina

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

utf-8 in html: caratteri di uso comune li metto in 1 byte. non so lo spazio che mi occupa, dipende dai caratteri che ci sono. ma in html non ci interessa la lunghezza della stringa
utf-16 in java

Se nella pagina web trovo ? o caratteri strani è perchè non ha dato info sul carattere, allora in browser interpreta da sè e quando sbaglia escono quei simboli casuali

Testo

- h1..h6
 - Titoli (heading) di parti del testo
- p
 - Paragrafo, unità di base per la suddivisione del testo
- b, i, ...
 - Formattazione del testo, (bold → grassetto), <i>(italic → corsivo)</i>, etc
 - Obsoleti (andrebbe usato CSS) ma mantenuti per compatibilità e semplicità
- br
 - HTML ignora spazi, tab, andate a capo, etc.
 - 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>

Qui c'è l'elenco delle varie
modalità di scrittura dei
caratteri speciali

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

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`
 - 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`

Immagini

- `img` – `src`, `alt`, `title`, `height`, `width`
 - *Non ha tag di chiusura*, tutte le informazioni sono negli attributi
 - **src**: l'indirizzo della risorsa, che può essere locale o meno
 - ``
 - ``
 - `alt`: testo alternativo, da mostrare se l'immagine non è accedibile
 - `title`: testo aggiuntivo mostrato quando il puntatore passa sull'immagine
 - ``
 - `height`, `width`: dimensioni dell'immagine
 - Specificandone una l'altra viene calcolata dal browser. Entrambe: l'immagine può essere distorta
 - Valore assoluto (pixel): ``
 - Percentuale sul viewport corrente: ``

Usate es per inserire la posizione

Gli dico la grandezza della mia pagina che darò in mano a googlemaps(per es) per farsi che si veda la mia posizione.

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&output=embed">
</iframe>
```

Con google scelgo un posto, condividi, incorporare mappa e lui mi dà il frame

Tabelle

Per inserire dati, non per organizzare visivamente la pagina

- 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

Blocco = div, inline = span

- Alcuni elementi implicano la creazione di un nuovo **blocco**, come h1..6, p, ul, li
- Altri, **inline**, sono considerati parte del blocco già esistente, come a, b, img
- L'elemento **div** rappresenta un blocco generico
- La sua controparte inline è **span**

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

Sarebbe un riquadro con dentro degli elementi (widget) per gestire l'interazione con gli utenti

- L'elemento **form** è uno tra i principali strumenti per gestire l'interazione con l'utente
- Permettono di inviare dati al sito web
- Il form contiene **widget** (elementi HTML visualizzati in modo standard), ognuno dei quali è usato per generare un parametro con i dati da inviare

Collegamento tra V e C
(tra View html e
Controller java), si
collegano via internet,
tramite protocollo HTTP
(protocollo per il
trasferimento di ipertesti
(= html))

Request – Response

- 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

Oppure HTTPs cioè
secure, è più criptato

Sono i 2 comandi utilizzati:
GET: voglio leggere una
risorsa
POST: voglio postare dei
cambiamenti

GET **macchina** su cui
vuoi fare get, (se non
specifico di default è sulla
macchina che lo richiede)
e **risorsa** che voglio
mandare (Come
funzionamento link) ?
parametri (a=12&b=27)

POST i parametri li metto
nel payload non sono
visibili (se https allora
criptati) es password o
dati riservati

Request:
V->C, C modifica M.
Poi
Response da C -> V

Siccome di parametri
sono visibile, posso
metterli in una history
(tipo tra i preferiti), e
posso avere richiamarlo il
più velocemente possibile
con quei parametri precisi

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)

```
<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>
```

Funziona solo se dentro al form

textarea puoi andare a capo, input no.
input non ha chiusura, textarea si

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>
```

input text (et al.) – textarea

- **input**

- *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:

- **text** (default) `<input type="text" name="user" value="Bob" maxlength="30" />`
- **password** (dati sensibili) `<input type="password" name="pwd" maxlength="30" required />`
- **hidden** (parametro nascosto) `<input type="hidden" name="invisible" value="notShown" />`
- **date** (scelta di un giorno) `<input type="date" name="milestone" />`

- **textarea**

- 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

```
<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

- 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, vengono associati al name 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>
```

select – option

- Scelta di una opzione da una lista a scomparsa
- Due controlli operano congiuntamente
 - **select** fa da container e definisce l'attributo **name**
 - **option** definisce il **value** per ogni singola voce
 - L'attributo **selected** specifica 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>
```

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 in produzione)

```
<head>
<!-- -->
<style>input {color: red;}</style>
</head>
```

```
input {color: red;}
```

./css/s27.css

```
<link rel="stylesheet"
      type="text/css"
      href="./css/s27.css"/>
```

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

Selettori

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

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

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

- Selezione degli elementi nella pagina a cui applicare la regola:
 - Tipo
 - Classe, attributo class
 - Identificatore, attributo id
 - Attributo
 - Pseudo classe (hover, checked, nth-child(), ...)
 - 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
- Le regole si applicano in cascata
- In generale, in caso di conflitto vince la regola più specifica

Selettori – esempi

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

```
<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**, ...
 - **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

```
<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%
}

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

th {
  background-color: lightblue;
}

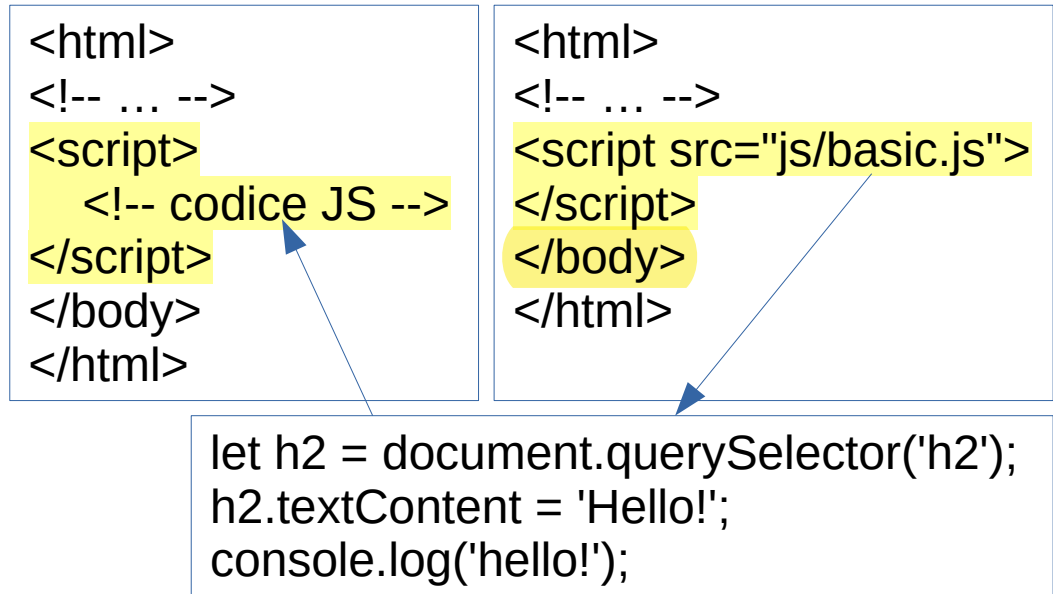
td {
  background-color: lightgreen;
}
```

JavaScript

- Linguaggio di programmazione interpretato, multi-paradigma, imperativo, funzionale, event-driven
- Nato nel 1995 per aggiungere funzionalità alla coppia HTML-CSS, è ora utilizzato un po' ovunque
- Dal 1997 ECMA ne coordina lo sviluppo, con il nome ufficiale di ECMAScript
- Nonostante il nome, è sostanzialmente diverso da Java

HTML – JavaScript

- Elemento `script`, subito prima della chiusura del `body`
- Il codice può essere:
 - Scritto direttamente nell'elemento `script` (sconsigliato in produzione)
 - Caricato da un file JS esterno, specificato nell'attributo `src`



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**)
- Non si esplicita il tipo, che può essere:
 - **string**: `let name = 'Bob';`
 - **number**: `let value = 42; // sia interi sia float`
 - **boolean**: `let flag = true;`
 - **object**: `let dog = { name : 'Zip', breed : 'Alsatian' };`
 - **array**: `let data = [1, 'Tom', false];`
- L'operatore **typeof()** ritorna il tipo dedotto da JS (o **undefined**)
- Per dichiarare costanti 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 postfixo)

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 di confronto

- Operatori che ritornano un booleano dal test ...
 - `===` di stretta uguaglianza (stesso tipo e valore)
 - `!==` di stretta disuguaglianza (diverso tipo o valore)
 - `<` il valore sulla sinistra è minore del valore sulla destra
 - `<=` il valore sulla sinistra è minore del valore sulla destra
 - `>` il valore sulla sinistra è maggiore del valore sulla destra
 - `>=` il valore sulla sinistra è maggiore del valore sulla destra
- Gli operatori non-strict `==` e `!=` vanno usati con cautela

Stringa

- Una stringa è una sequenza di caratteri delimitata da apici singoli o doppi
- Per concatenare stringhe si usa 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 substr: `s.indexOf(sub)` // `-1` not found
- Estrazione di substr: `s.slice(i)`, `s.slice(i, j)`
- 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
- 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]);
}
```

Da array a stringa con
`join`

```
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`;

Funzioni

- Definizione di una funzione

```
function f() {  
}
```

- Invocazione di una funzione

```
f();
```

- Una funzione può essere anonima, ed essere assegnata ad una variabile

```
let x = function() {  
}
```

- Una funzione con parametri che ritorna un valore

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

AJAX e XMLHttpRequest

All'interno della pagina

- **Asynchronous JavaScript And XML**
- 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 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>
```