

PER ALTRI APPUNTI CONSULTARE IL SITO:
https://luigi-v.github.io/Appunti_Universita/

URL

Ogni risorsa sul web ha un **indirizzo unico**, nel formato **URL (Uniform Resource Locators)**.

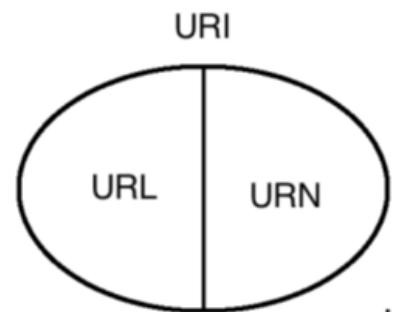
Gli URL sono un particolare tipo di **URI (Uniform Resource Identifier)**, stringa di caratteri, che forniscono un meccanismo per identificare una risorsa.

Caratteristiche di un **URI**:

- È un **concetto generale**: non fa riferimento necessariamente a risorse accessibili tramite HTTP o ad entità disponibili in rete (quando lo fa, diventa URL).
- È un **mapping concettuale ad una entità**: non si riferisce necessariamente ad una particolare versione dell'entità esistente in un dato momento.

Esistono due specializzazioni del concetto di **URI**:

- **Uniform Resource Name (URN)**: identifica una risorsa per mezzo di un "nome" che deve essere unico e restare valido anche se la risorsa diventa non disponibile o cessa di esistere.
- **Uniform Resource Locator (URL)**: identifica una risorsa per mezzo della locazione nella rete.



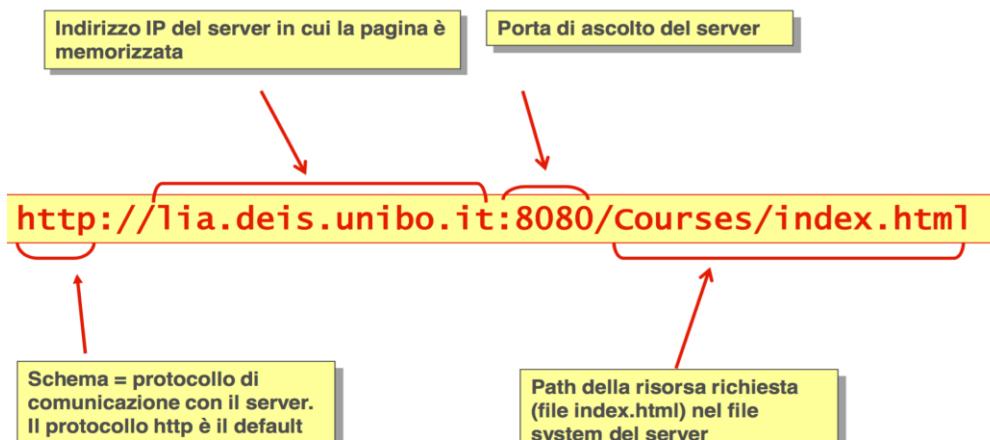
Un **URL** tiene conto anche della modalità per accedere alla risorsa:

- Specifica il protocollo necessario per il trasferimento della risorsa stessa (non solo HTTP, quindi...)
- La prima parte corrisponde al protocollo utilizzato
- La parte rimanente del nome dipende dal protocollo

Nella sua forma più comune (**schema HTTP-like**) la sintassi è

<protocol>://[<username>:<password>@]<host>[:<port>][/<path>[?<query>][#fragment]]

- **<protocol>**: Describe il protocollo da utilizzare per l'accesso al server (HTTP, HTTPS, FTP, ...);
- **<username>:<password>@**: credenziali per l'autenticazione;
- **<host>**: indirizzo server su cui risiede la risorsa. Può essere un indirizzo IP logico o fisico (www.unisa.it o 193.205.160.20);
- **<port>**: definisce la porta da utilizzare (TCP protocollo di trasporto per http, 80 default);
- **<path>**: percorso (pathname) che identifica la risorsa nel file system del server (Home page della pagina se manca);
- **<query>**: una stringa di caratteri che consente di passare al server uno o più parametri. Di solito ha questo formato: **parametro1=valore¶metro2=valore2...**;
- **fragment**: è una breve stringa di caratteri che si riferisce a una risorsa che è subordinata a un'altra risorsa primaria.



HTTP

HTTP è l'acronimo di **HyperText Transfer Protocol**, è il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) da **server** a **client**.

Gestisce sia le **richieste (URL)** inviate al server che le **risposte (pagine)** inviate al client, è un protocollo **stateless**, ovvero *né il server né il client mantengono informazioni relative ai messaggi scambiati*.

TERMINOLOGIA:

Client: programma applicativo che stabilisce una connessione al fine di inviare delle richieste;

Server: programma applicativo che accetta connessioni al fine di ricevere richieste ed inviare specifiche risposte con le risorse richieste;

Connessione: circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione;

Messaggio: è l'unità base di comunicazione HTTP, è definita come una specifica sequenza di byte concettualmente atomica:

- **Request:** messaggio HTTP di **richiesta** (*Client -> Server*)
- **Response:** messaggio HTTP di **risposta** (*Server -> Client*)

Resource: oggetto di tipo dato univocamente definito;

URI: Uniform Resource Identifier – identificatore unico per una risorsa;

Entity: rappresentazione di una risorsa, può essere incapsulata in un messaggio, tipicamente di risposta;

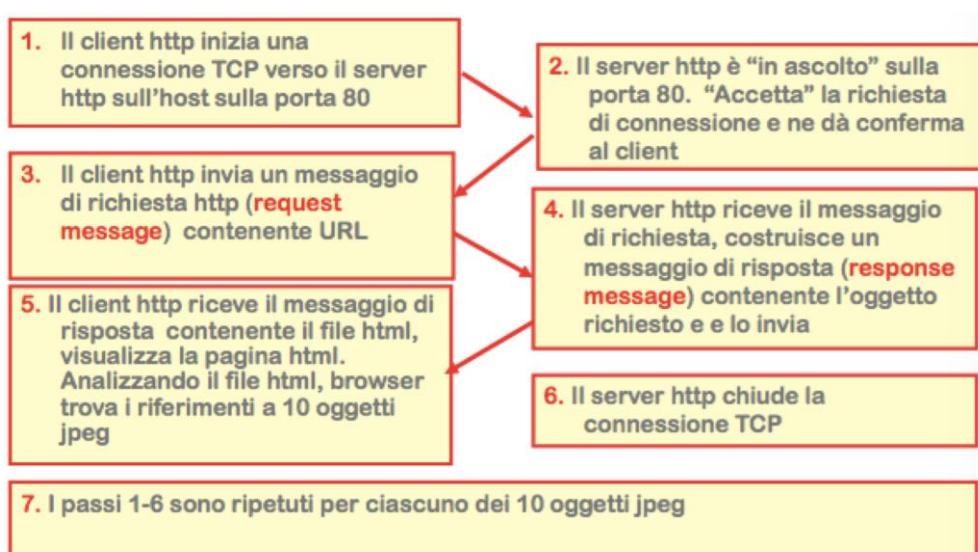
Una **risorsa** può essere una pagina X, mentre una sua corrispondente **entità** è il testo contenuto in essa. Se cambia il testo la risorsa resta comunque la pagina X.

TIPI DI CONNESSIONE

Sia le richieste al server che le risposte ai client sono trasmesse usando stream **TCP**. Segue uno schema di questo tipo:

- **server** rimane in ascolto (server passivo), tipicamente sulla porta 80
- **client** apre una connessione **TCP** sulla porta 80
- **server** accetta la connessione (possibili più connessioni in contemporanea?)
- **client** manda una richiesta
- **server** invia la risposta e chiude la connessione

Ipotizziamo di volere richiedere una pagina composta da un file HTML e 10 immagini JPEG:



La stessa connessione HTTP può essere utilizzata per una serie di richieste e una serie corrispondente di risposte.

La differenza principale tra HTTP 1.0 e 1.1 è la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione.

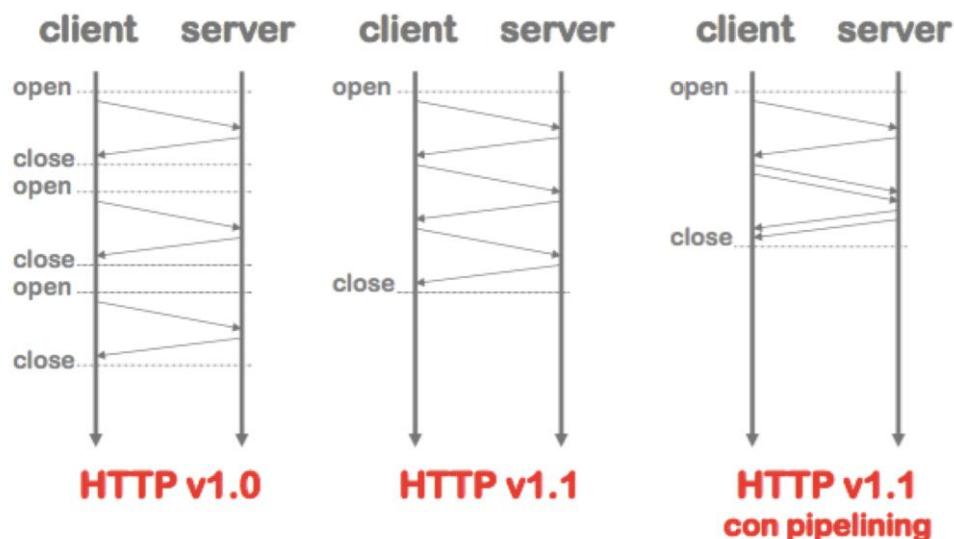
Le connessioni 1.0 vengono dette **non persistenti** mentre quelle 1.1 vengono definite **persistenti**.

Il server lascia aperta la connessione TCP dopo aver spedito la risposta e può quindi ricevere le richieste successive sulla stessa connessione.

Il server HTTP chiude la connessione quando viene specificato nell'header del messaggio (specificata dal **client**) oppure quando non è usata da un certo tempo (**time out**).

HTTP V1.1 E PIPELINING

Il pipelining consiste nell'invio di **molteplici richieste da parte del client prima di terminare la ricezione delle risposte**. Le risposte debbono però essere date nello stesso ordine delle richieste



Metodi dei protocolli HTTP:

Una richiesta è un nome di metodo http che indica al server il tipo di richiesta e come deve essere formato il resto del messaggio.

Differenze tra...

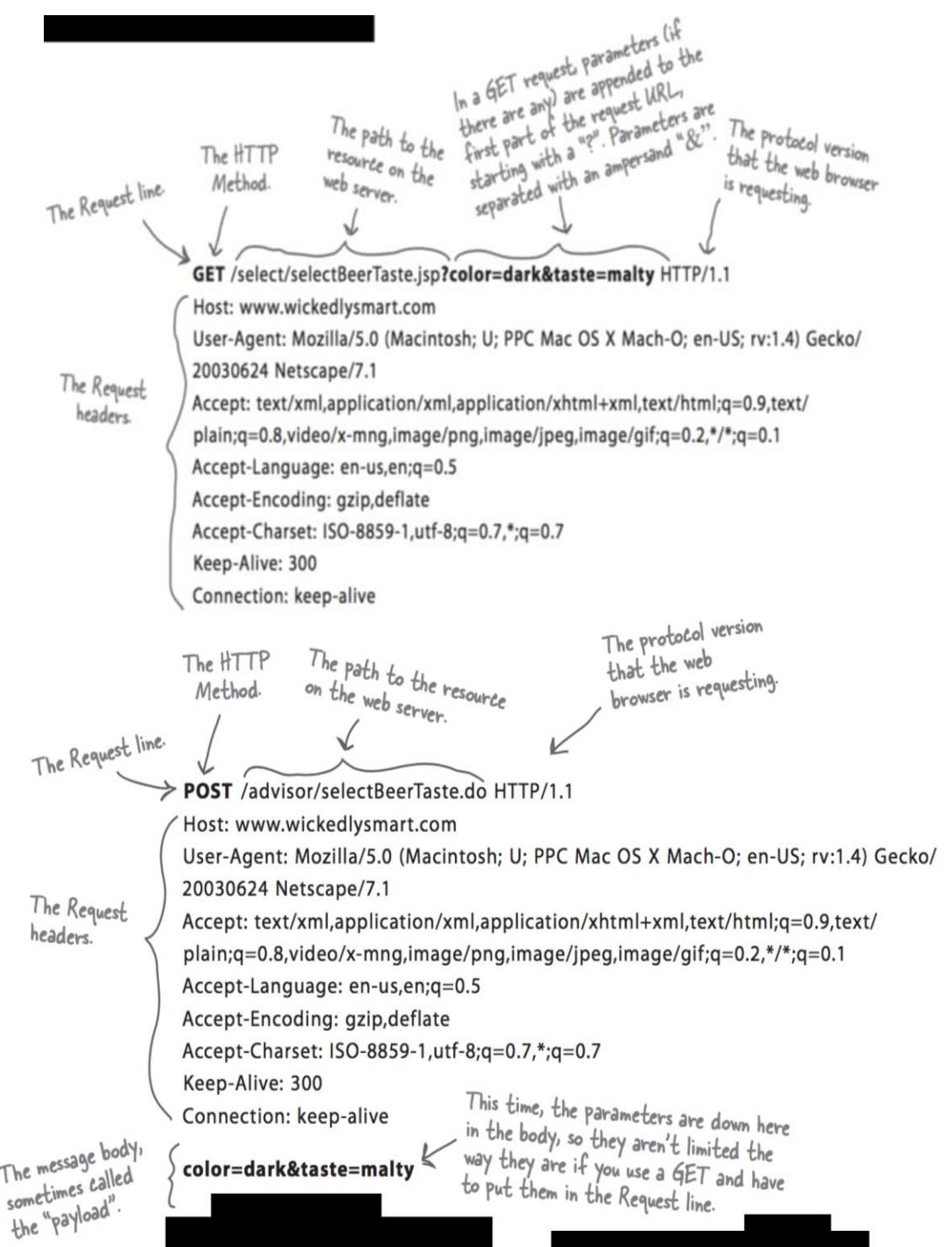
- GET:** richiede una risorsa ad un server, il metodo più frequente è il click sui link su un documento o specificando un URL. La quantità di caratteri è limitata dalla lunghezza massima di un URL, e i dati inviati con *GET* vengono aggiunti all'URL nella barra del browser, e quindi i **dati sono esposti**.
- POST:** è possibile richiedere qualcosa e allo stesso tempo inviare i dati del modulo al server, i dettagli della risorsa sono del *body del messaggio* e no nell'URL. **Non ci sono limiti di lunghezza nei parametri.**

GET POST

Indicato per pochi parametri	Indicato per grandi quantità di dati
Solo parametri testuali	Dati in qualsiasi formato (testi, immagini, video)
Possibilità di inserire un URL con parametri opportuni all'interno di una pagina HTML	Solo in risposta ad una form

es.

...Ricerca Altavista sulla complessità ...



- **PUT:** Chiede la memorizzazione sul server di una risorsa all'URL specificato, serve per trasmettere delle informazioni dal client al server. A differenza del POST però si ha la creazione di una risorsa. L'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET con lo stesso nome in seguito.
- **DELETE:** Richiede la cancellazione della risorsa riferita dall'URL specificato.
 - Sono normalmente disabilitati sui server pubblici(PUT & DELETE)
- **HEAD:** simile al GET, ma il server deve rispondere soltanto con gli header relativi, senza body. Viene usato per verificare un URL.
- **OPTIONS:** serve per richiedere informazioni sulle opzioni disponibili per la comunicazione.
- **TRACE:** invoca il loop-back remoto del messaggio di richiesta, ovvero consente al client di vedere che cosa è stato ricevuto dal server(diagnostica e testing).

HTTP RESPONSE

Describe le informazioni indicate dal protocollo del browser, se la richiesta ha avuto successo e che tipo di contenuto è incluso nel body(es HTML).

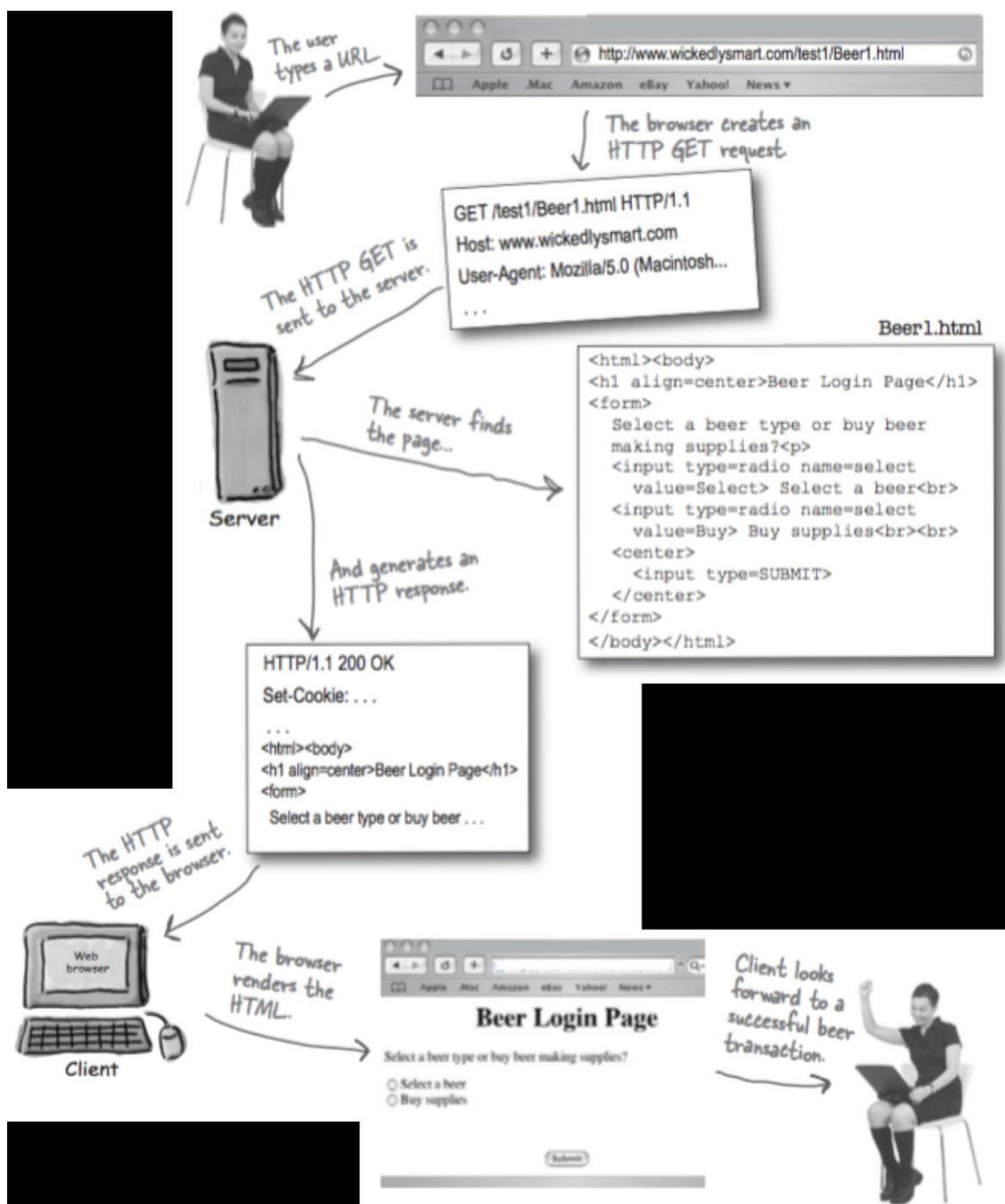
MIME MULTIPURPOSE INTERNET MAIL EXTENSIONS (MIME)

Usato per specificare al browser la forma di un file inviato dal server, inserito all'inizio del documento. Specificando il tipo o sottotipo del file(es text, image...)

CODICI DI STATO

Lo **status code** è un numero di tre cifre, di cui la prima indica la classe della risposta e le altre due la risposta specifica:

- **1xx: Informational.** Una risposta temporanea alla richiesta(es 100 Continue= se il client non ha ancora mandato il body);
- **2xx: Successful.** Il server ha ricevuto la richiesta(200 Ok= GET con successo, 201 Created =PUT con successo);
- **3xx: Redirection.** Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client(301 Moved permanently =URL non valida);
- **4xx: Client error.** La richiesta del client non può essere soddisfatta per un errore da parte del client (400 Bad request=errore sintattico o richiesta non autorizzata).
- **5xx: Server error.** La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema(501 Not implemented =metodo non conosciuto dal server).



COOKIE

Parallelamente alle sequenze **request/response**, il protocollo prevede una struttura dati che si muove come un token, dal client al server e viceversa.

I **cookie** possono essere generati sia dal client che dal server. Dopo la loro creazione vengono sempre passati ad ogni trasmissione di **request** e **response**.

Hanno come scopo quello di fornire un supporto per il **mantenimento di stato** in un protocollo come HTTP, che è essenzialmente **stateless**.

Un "cookie" può essere impostato dal lato-server e inviato ai client utilizzando sia le Servlet che le JSP.

Le informazioni dei cookie sono memorizzate su disco del computer locale.

STRUTTURA DEI COOKIE

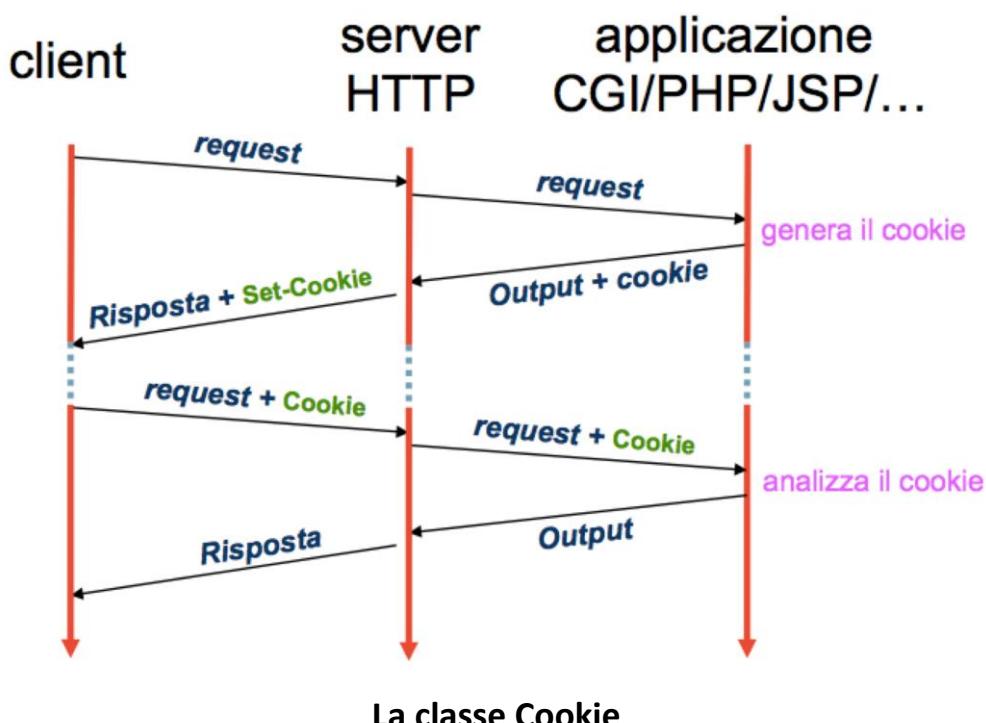
I cookie sono una collezione di stringhe:

- **Key:** identifica univocamente un cookie all'interno di un **domain:path**
- **Value:** valore associato al cookie (è una stringa di max 255 caratteri)
- **Path:** posizione nell'albero di un sito al quale è associato (di default /)
- **Domain:** dominio dove è stato generato
- **Max-age:** (*opzionale*) numero di secondi di vita
- **Secure:** (*opzionale*) non molto usato. Questi cookie vengono trasferiti se il protocollo è sicuro (**HTTPS**)
- **Version:** identifica la versione del protocollo di gestione dei cookie

HEADER DEI COOKIE

I cookies usano due header, uno per la **risposta**, ed uno per **richieste** successive:

- **Set-Cookie:** header della risposta, il client può memorizzarlo e rispedirlo alla prossima richiesta
- **Cookie:** header della richiesta. Il client decide se spedirlo sulla base del nome del documento, dell'indirizzo IP del server, e dell'età del cookie



Un cookie contiene un certo numero di informazioni, tra cui:

- una **coppia** nome/valore (Caratteri non utilizzabili: [] () = , " / ? @ : ;)
- il **dominio Internet** dell'applicazione che ne fa uso

- **path** dell'applicazione
- una **expiration date** espressa in secondi (0 o numero negativo indica che il cookie non sarà memorizzato, 60*60*24 indica 24 ore)
- un **valore booleano** per definirne il livello di sicurezza

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>

La **classe Cookie** modella il cookie HTTP:

- Si recuperano i cookie dalla **request** utilizzando il metodo **getCookies()**
- Si aggiungono cookie alla **response** utilizzando il metodo **addCookie()**

Con il metodo **setSecure(true)** il client viene forzato a inviare il cookie solo su protocollo sicuro (**HTTPS**)

creazione

```
Cookie c = new Cookie("MyCookie", "test");
c.setSecure(true);
c.setMaxAge(-1);
c.setPath("/");
response.addCookie(c);
```

lettura

```
Cookie[] cookies = request.getCookies();
if(cookies != null)
{
    for(int j=0; j<cookies.length(); j++)
    {
        Cookie c = cookies[j];
        out.println("Un cookie: " +
                    c.getName()+"="+c.getValue());
    }
}
```

Per eliminare un cookie, è sufficiente seguire i seguenti tre passaggi:

- Leggere un cookie già esistente e memorizzarlo nell'oggetto Cookie
- Imposta l'età del cookie a zero utilizzando il metodo **setMaxAge()**
- Aggiungi questo cookie nell'intestazione della risposta

```
Cookie[] cookies = null;
cookies = request.getCookies();
Cookie cookie = cookies[i] //i-esimo Cookie
cookie.setMaxAge(0);
response.addCookie(cookie);
```

È un linguaggio di **markup** ('contrassegno' o 'di marcatura'), che permette di indicare come disporre gli elementi all'interno di una pagina grazie ad appositi marcatori, detti **tag** ('etichette').

HTML, serve a definire quali sono gli elementi in gioco, stabilire collegamenti (link) tra le pagine e l'importanza che hanno i testi, creare form per gli utenti, fissare titoli, caricare immagini, video, etc.

```
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
</head>
<body>
  <h1>My First Heading</h1>
  <p>My first paragraph.</p>
</body>
</html>
```

- La dichiarazione **<!DOCTYPE html>** rappresenta il tipo di documento e aiuta i browser a visualizzare correttamente le pagine Web.
- L' elemento **<html>** è l'elemento principale di una pagina HTML
- L' elemento **<head>** contiene metadati sul documento
- L' elemento **<title>** specifica un titolo per il documento
- L' elemento **<body>** contiene il contenuto della pagina visibile
- L' elemento **<h1>** definisce una grande intestazione
- L' elemento **<p>** definisce un paragrafo

I **tag HTML** sono nomi di elementi circondati da parentesi angolari:

- I tag HTML vengono normalmente in coppie come **<p>** e **</p>**
- Il primo tag in una coppia è il **tag iniziale**, il secondo tag è il **tag finale**
- Il tag di chiusura è scritto come il tag di inizio, ma con una **barra di avanzamento** inserita prima del nome del tag

Nello standard HTML5, il <html> tag, il <body> tag e il <head> tag possono essere omessi.

Caricare la pagina significa acquisirne il contenuto, più tecnicamente il browser richiede che venga effettuato un **trasferimento di file**. La pagina HTML è raggiungibile facendo riferimento ad un certo indirizzo, effettuando richieste:

- “**locale**”, quando i file si trovano sul device o sul computer su cui gira il browser.
file:///D:/HTML.it/GUIDE/HTML/introduzione.html
- “**remoto**”, quando i file sono su un server da contattare tramite internet (o altra rete).
http://www.html.it/guida-html/introduzione.html

È importante minimizzare quanto più possibile i **tempi di caricamento**.

Il **rendering** della pagina è la fase in cui il browser interpreta i documenti HTML e dispone sullo schermo gli elementi (testi, immagini, filmati) a seconda delle indicazioni contenute.

Struttura della pagina HTML

```
<Html>
  <Head>
    <title> Titolo pagina </ title>
  </ Head>

  <Body>
    <h1> Questa è un'intestazione </ h1>
    <p> Questo è un paragrafo. </ p>
    <p> Questo è un altro paragrafo. </ p>
  </ Body>
</ Html>
```

1. Tutti i documenti HTML devono iniziare con una dichiarazione del tipo di documento: `<!DOCTYPE html>`
2. Il documento HTML stesso inizia `<html>` e finisce con `</html>`
3. La parte visibile del documento HTML si trova tra `<body>` e `</body>`

Elementi - https://www.w3schools.com/html/html_elements.asp

Un elemento HTML è un contenitore e il suo contenuto è delimitato da tag di apertura e di chiusura.

`< tagname > Il contenuto va qui ... < / tagname >`

Gli elementi HTML senza contenuto sono chiamati elementi vuoti, non hanno un tag di fine, come l'elemento `
`.

Elementi Block - https://www.w3schools.com/html/html_blocks.asp

Un elemento a livello di blocco `<div>` inizia sempre su una nuova riga e occupa l'intera larghezza disponibile (si estende da sinistra verso destra).

`<div>Hello</div>`
`<div>World</div>` https://www.w3schools.com/html/tryit.asp?filename=tryhtml_block_div

L'elemento `<div>` è spesso usato come contenitore per altri elementi HTML, non ha attributi obbligatori, ma **style**, classe id sono molto comuni se utilizzato insieme al CSS, creando diversi stili di contenuto.

`<div style="background-color:black;color:white;padding:20px;">`

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_div_capitals

Elementi Block in HTML:

`<address>` `<blockquote>` `<dd>` `<div>` `<dl>` `<dt>` `<fieldset>` `<footer>` `<form>`
`<h1> - <h6>` `<header>` `<hr>` `` `` `<p>` `<pre>` `<section>` `<table>` ``

Elementi inline - https://www.w3schools.com/html/html_blocks.asp

Un elemento in linea `` non inizia su una nuova riga e occupa solo la larghezza necessaria.

`Hello`
`World` https://www.w3schools.com/html/tryit.asp?filename=tryhtml_inline_span

L'elemento **** è spesso usato come contenitore per del testo, non ha attributi obbligatori, ma style, classe id sono comuni se utilizzato insieme al CSS, utilizzato per lo stile di parti del testo.

<h1>My Important Heading</h1>

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_span_red

Elementi Inline in HTML:

```
<a>   <abbr>    <b>    <big>    <button>    <cite>    <code>    <dfn>    <em>    <i>    <img>
<input>    <label>    <object>    <output>    <q>    <script>    <select>    <small>    <span>
<strong>    <sub>    <sup>    <textarea>    <time>    <var>
```

Attributi - https://www.w3schools.com/html/html_attributes.asp

Gli attributi forniscono informazioni aggiuntive sugli elementi HTML. Gli attributi forniscono **informazioni aggiuntive** su un elemento, e sono sempre specificati nel **tag di inizio**.

Gli attributi di solito vengono in coppie nome / valore come: **name = "value"**.

L'attributo href (link o collegamenti)

I collegamenti HTML sono definiti con il tag **<a>**. L'indirizzo del link è specificato nell'attributo **href**.

La sua sintassi è : Link text

Visit our HTML tutorial

Un **link locale** (link allo stesso sito web) è specificato con un URL relativo (senza https:// www).

HTML Images

Link: l'attributo di destinazione

L'attributo **target** specifica dove aprire il documento collegato, e può avere uno dei seguenti valori:

- **_blank** - Apre il documento collegato in una *nuova finestra* o scheda
- **_self** - Apre il documento collegato nella *stessa finestra* / scheda di dove è stato fatto clic
- **_parent** - Apre il documento collegato nel *frame principale*
- **_top** - Apre il documento collegato nel *corpo completo della finestra*
- *Framename* - apre il documento collegato in una cornice con nome

Visit W3Schools!

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_target

Link - Immagine come collegamento

```
<a href="default.asp">
  
</a>
```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_image

Link Titoli

L'attributo **title** specifica informazioni extra su un elemento. Le informazioni vengono spesso visualizzate come testo di descrizione quando il mouse si sposta sull'elemento.

Visit our HTML Tutorial

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_links_title

L'attributo src

Le immagini HTML sono definite con il tag ****.

Il nome file dell'immagine sorgente è specificato nell'attributo **src**.

Le immagini in HTML hanno un set di attributi **dimensione**, che specifica la larghezza e l'altezza dell'immagine.

```

```

L'attributo **alt** specifica un testo alternativo da utilizzare, quando un'immagine non può essere visualizzata

Il valore dell'attributo può essere letto dagli screen reader.

```

```

L'attributo di stile

L'attributo **style** è usato per specificare lo stile di un elemento, come colore, carattere, dimensione, ecc.

```
<p style="color:red">I am a paragraph</p> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_attributes\_style
```

L'attributo lang

La lingua del documento può essere dichiarata nel <html> tag, dichiarata con l'attributo **lang**.

```
<html lang="en-US">
```

L'attributo titolo (suggerimento)

Il valore dell'attributo titolo verrà visualizzato come suggerimento quando si passa il mouse sopra.

```
<p title="I'm a tooltip">
```

This is a paragraph.

```
</p>
```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_attributes_title

L'attributo di classe - https://www.w3schools.com/html/html_classes.asp

L'attributo **class** viene utilizzato per definire stili uguali per elementi con lo stesso nome di classe.

Pertanto, tutti gli elementi HTML con lo stesso attributo **class** avranno lo stesso formato e stile.

```
<head>
<style>
.cities {
background-color: black;
color: white;
margin: 20px;
padding: 20px;
}
</style>
</head>
<body>
<div class="cities">
<h2>London</h2>
<p>London is the capital of England.</p>
</div>

<div class="cities">
<h2>Paris</h2>
<p>Paris is the capital of France.</p>
</div>
</body> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_classes\_capitals
```

L'attributo **class** può essere utilizzato anche su *elementi in linea*.

```

<style>
  .note {
    font-size: 120%;
    color: red;
  }
</style>
</head>
<body>
  <h1>My <span class="note">Important</span> Heading</h1>
  <p>This is some <span class="note">important</span> text.</p>
</body>      https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_classes\_span

```

Nota: il nome della classe è case sensitive!

Tag diversi possono condividere la stessa classe

Tag diversi, come **<h2>** e **<p>**, possono avere lo stesso nome di classe e quindi condividere lo stesso stile

```

<h2 class="city">Paris</h2>
<p class="city">Paris is the capital of France</p>

```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_classes_tags

L'attributo id - https://www.w3schools.com/html/html_id.asp

L'attributo **id** specifica un ID univoco per un elemento HTML (il valore deve essere univoco all'interno del documento HTML).

Il valore id può essere utilizzato da CSS e JavaScript per eseguire determinate attività per un elemento univoco con il valore id specificato.

```

<style>
#myHeader {
  background-color: lightblue;
  color: black;
  padding: 40px;
  text-align: center;
}
</style>
<h1 id="myHeader">My Header</h1> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_id\_css

```

Nota: il valore id è sensibile al maiuscolo / minuscolo.

Differenza tra classe e ID

Un elemento HTML può avere solo un ID univoco che appartiene a quel singolo elemento, mentre un nome di classe può essere utilizzato da più elementi. https://www.w3schools.com/html/tryit.asp?filename=tryhtml_id_class

Intestazioni - https://www.w3schools.com/html/html_headings.asp

<h1>definisce la voce più importante. <h6>definisce l'intestazione meno importante.

```

<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>

```

Il tag **<hr>** definisce un'interruzione tematica in una pagina HTML e viene spesso visualizzata come una riga orizzontale. https://www.w3schools.com/html/tryit.asp?filename=tryhtml_headings_hr

Paragrafi - https://www.w3schools.com/html/html_paragraphs.asp

Con HTML, non puoi modificare l'output aggiungendo spazi extra o linee extra nel tuo codice HTML.

Il browser rimuoverà spazi aggiuntivi e linee extra quando viene visualizzata la pagina.

```
<p>This is a paragraph.</p>
```

```
<p>This is another paragraph.</p>
```

L'elemento **
** HTML definisce **un'interruzione di riga**.

```
<p>This is<br>a paragraph<br>with line breaks.</p>
```

L'elemento **<pre>** HTML definisce il testo preformatto.

Il testo all'interno di un elemento **<pre>** viene visualizzato in un font a larghezza fissa e conserva spazi e interruzioni di riga.

```
<pre>
```

```
    My Bonnie lies over the ocean.
```

```
    Oh, bring back my Bonnie to me.
```

```
</pre>
```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_pre

Commenti - https://www.w3schools.com/html/html_comments.asp

Puoi aggiungere commenti al tuo sorgente HTML usando la seguente sintassi:

```
<!-- Write your comments here -->
```

I commenti non vengono visualizzati dal browser, ma possono aiutare a documentare il codice sorgente.

Stili - https://www.w3schools.com/html/html_styles.asp

L'impostazione dello stile di un elemento HTML può essere eseguita con l'attributo **style**.

L'attributo **style** HTML ha la seguente sintassi : **<tagname style="property:value;">**

Le **proprietà** sono:

- **background-color** definisce il colore di sfondo per un elemento HTML.

```
<body style="background-color:pink;">  
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_styles\_background-color
```

- **color** definisce il colore del testo per un elemento HTML.

```
<h1 style="color:red;">This is a heading</h1>  
<p style="color:blue;">This is a paragraph.</p>  
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_styles\_color
```

- **font-family** definisce il carattere da utilizzare per un elemento HTML.

```
<h1 style="font-family:verdana;">This is a heading</h1>  
<p style="font-family:courier;">This is a paragraph.</p>  
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_styles\_font-family
```

- **font-size** definisce la dimensione del testo per un elemento HTML.

```
<h1 style="font-size:300%;">This is a heading</h1>  
<p style="font-size:160%;">This is a paragraph.</p>  
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_styles\_font-size
```

- **text-align** definisce l'allineamento orizzontale del testo per un elemento HTML.

```
<h1 style="text-align:center;">Centered Heading</h1>
<p style="text-align:center;">Centered paragraph.</p>
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_styles\_text-align
```

- **border** definisce il bordo ed il suo colore.

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_color\_border
```

Formattazione del testo - https://www.w3schools.com/html/html_formatting.asp

HTML usa elementi come **** e **<i>** per la formattazione dell'output, come il testo in **grassetto** o in **corsivo**.

L'elemento **** definisce il testo in **grassetto**, senza alcuna importanza aggiuntiva.

```
<b>This text is bold</b> = This text is bold.
```

L'elemento **** definisce il testo forte, con un'importanza "forte" semantica aggiunta.

```
<strong>This text is strong</strong> = This text is strong.
```

L'elemento **<i>** definisce il testo in **corsivo**, senza alcuna importanza aggiuntiva.

```
<i>This text is italic</i> = This text is italic.
```

L'elemento **** definisce il testo enfatizzato, con un'importanza semantica aggiunta.

```
<em>This text is emphasized</em> = This text is emphasized.
```

L'elemento **<small>** definisce un testo più piccolo.

```
<h2>HTML <small>Small</small> Formatting</h2> = HTML Small Formatting
```

L'elemento **<mark>** definisce segnato o evidenziato testo.

```
<h2>HTML <mark>Marked</mark> Formatting</h2> =
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_formatting\_mark
```

L'elemento **** definisce cancellato (rimosso) testo.

```
<p>My favorite color is <del>blue</del> red.</p> = My favorite color is blue red.
```

L'elemento **<ins>** definisce sottolineato (aggiunto) testo.

```
<p>My favorite <ins>color</ins> is red.</p> = My favorite color is red.
```

L'elemento **<sub>** definisce il testo con indice.

```
<p>This is <sub>subscripted</sub> text.</p> = This is subscripted text.
```

L'elemento **<sup>** definisce il testo in apice.

```
<p>This is <sup>superscripted</sup> text.</p> = Questo è un testo in apice.
```

L'elemento **<q>** definisce una breve citazione, tra **virgolette**.

```
<p>WWF's goal is to: <q>Build a future</q></p> = WWF's goal is to: "Build a future"
```

L'elemento **<blockquote>** definisce una sezione citata da un'altra fonte.

```
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_formatting\_blockquote
```

L'elemento **<abbr>** definisce un'abbreviazione o un acronimo.

```
https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_formatting\_abbr
```

L'elemento **<address>** è visualizzato in corsivo ed aggiungerà un'interruzione di riga prima e dopo.

L'elemento <**cite**> definisce il titolo di un'opera.

Tabelle - https://www.w3schools.com/html/html_tables.asp

Una tabella HTML è definita con il tag <**table**>, e può avere diversi figli:

- <**tr**>, definisce la **riga** della tabella
- <**th**>, definisce l'intestazione (**header**) della tabella, scritte di default in grassetto e centrate
- <**td**>, definisce i **dati** o celle all'interno della tabella, possono contenere tutti i tipi di elementi HTML

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
  </tr>
</table> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_table
```

Celle che si estendono su molte colonne

Per fare in modo che una cella si estenda su più di una colonna, usa l'attributo **colspan**.

```
<table>
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_table\_colspan
```

Celle che si estendono su più righe

Per fare in modo che una cella si estenda su più di una riga, usa l'attributo **rowspan**.

```
<table>
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
```

```
</tr>
</table> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_table\_rowspan
```

Aggiunta di una didascalia

Per aggiungere una didascalia a una tabella, usa il tag **<caption>** (*immediatamente dopo il <table>*).

```
<table>
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_tables2
```

Lista - https://www.w3schools.com/html/html_lists.asp

Una **lista non ordinata(unordered)** inizia con il tag ****. Ogni elemento della lista inizia con il tag ****.

Gli elementi dell'elenco saranno *contrassegnati con piccoli cerchi neri* per impostazione predefinita.

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
</ul> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_lists\_unordered
```

Una **lista ordinata(ordered)** inizia con il tag ****. Ogni elemento della lista inizia con il tag ****.

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
</ol> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_lists\_ordered
```

Una **lista di descrizione** è una lista di termini, con una descrizione di ciascun termine, può avere figli:

- **<dl>**, definisce la **lista di descrizione**
- **<dt>**, definisce il **titolo** dell'elemento elencato
- **<dd>**, definisce la **descrizione** dell'elemento elencato

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_lists\_description
```

Controllo della lista (ordered list)

Per impostazione predefinita, un elenco ordinato inizierà il conteggio da 1. Se si desidera iniziare il conteggio da un numero specificato, è possibile utilizzare l'attributo **start**.

```
<ol start="50">
  <li>Coffee</li>
```

```
<li>Tea</li>
<li>Milk</li>
</ol>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_lists\_start
```

Attributo Type (ordered list)

L'attributo **type** del tag ****, definisce il tipo del marcatore dell'elemento della lista:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

Indicatore dell'elenco (unordered list)

La proprietà CSS **list-style-type** viene utilizzata per definire lo stile del marcatore dell'elemento della lista.

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_lists_unordered_disc

Entità - https://www.w3schools.com/html/html_entities.asp

I **caratteri riservati** in HTML devono essere sostituiti con entità carattere, anche quelli che non sono presenti sulla tastiera possono essere sostituiti da entità.

Alcuni caratteri sono riservati in HTML, infatti se si utilizzano i segni meno di (<) o maggiore di (>) nel testo, il browser potrebbe mescolarli con tag.

Un'entità di carattere comune utilizzata in HTML è lo spazio senza interruzione:

Uno spazio senza interruzione è uno spazio che non si spezzerà in una nuova linea.

Result	Description	Entity Name	Entity Number
	non-breaking space	 	
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
"	double quotation mark	"	"
'	single quotation mark (apostrophe)	'	'
¢	cent	¢	¢
£	pound	£	£
¥	yen	¥	¥
€	euro	€	€
©	copyright	©	©
®	registered trademark	®	®

Nota: i nomi delle entità fanno distinzione tra maiuscole e minuscole.

L'elemento <form> - https://www.w3schools.com/html/html_forms.asp

Definisce un modulo che viene utilizzato per raccogliere l'input dell'utente:

```
<form>
  .
  form elements
  .
</form>
```

Un modulo HTML contiene **elementi del modulo** che sono diversi tipi di elementi di input, come campi di testo, caselle di controllo, pulsanti di opzione, pulsanti di invio e altro.

L'elemento <input>

L'elemento <input> può essere visualizzato in diversi modi, a seconda dell'attributo **type**:

<input type = "text"> Definisce un campo di immissione del **testo** su una riga

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_text

<input type = "radio"> Definisce un **pulsante di opzione** (per selezionare una delle molte scelte)

https://www.w3schools.com/html/html_forms.asp

<input type = "submit"> Definisce un **pulsante di invio** dei dati del modulo a un **gestore di moduli**

Il **form-handler** è in genere una pagina del server con uno script per l'elaborazione dei dati di input.

Il gestore di moduli è specificato nell'attributo di **action** del modulo.

L'attributo **action** definisce l'azione da eseguire quando viene inviato il modulo.

<form action="/action_page.php">

First name:

<input type="text" name="firstname" value="Mickey">


```
<input type="submit" value="Submit">  
</form> https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_form\_submit
```

Se l'attributo **action** è omesso, l'azione viene impostata sulla pagina corrente.

Se l'attributo **type** è omesso, il campo di input ottiene il tipo predefinito: "text".

L'attributo target

L'attributo **target** specifica se il risultato inviato verrà aperto in una nuova scheda del browser, in una cornice o nella finestra corrente.

```
<form action="/action_page.php" target="_blank">
```

- **_blank** - Apre il documento collegato in una *nuova finestra* o scheda
- **_self (predefinito)** - Apre il documento collegato nella *stessa finestra / scheda* di dove è stato fatto clic
- **_parent** - Apre il documento collegato nel *frame principale*
- **_top** - Apre il documento collegato nel *corpo completo della finestra*
- **Framename** - apre il documento collegato in una cornice con nome

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_target

L'attributo method

L'attributo **method** specifica il metodo HTTP (**GET** o **POST**) da utilizzare quando si inviano i dati del modulo.

```
<form action="/action_page.php" method="get">
```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_post

Il metodo predefinito quando si inviano i dati del modulo è **GET**, i dati del modulo inviato saranno **visibili nel campo dell'indirizzo della pagina**: /action_page.php?firstname=Mickey&lastname=Mouse

- Aggiunge i dati di form nell'URL in coppie nome / valore
- La lunghezza di un URL è limitata (circa 3000 caratteri)
- Non utilizzare mai GET per inviare dati sensibili! (sarà visibile nell'URL)
- Utile per l'invio di moduli in cui un utente desidera aggiungere un segnalibro al risultato
- GET è migliore per i dati non protetti, come le stringhe di query in Google

Utilizzare sempre **POST** se i dati del modulo contengono informazioni sensibili o personali. Il metodo **POST** non visualizza i dati del modulo inviato nel campo dell'indirizzo della pagina.

- Il POST non ha limiti di dimensioni e può essere utilizzato per inviare grandi quantità di dati.
- Non è possibile aggiungere ai segnalibri moduli con POST

L'attributo name

Ogni campo di input deve avere un attributo name da inviare, se è omesso, i dati di quel campo di input non verranno inviati affatto.

```
Last name:<br><input type="text" name="lastname" value="Mouse"><br><br>
```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_submit_id

Raggruppamento dei dati del modulo con <fieldset>

L'elemento **<fieldset>** viene utilizzato per raggruppare i dati correlati in un modulo.

L'elemento **<legend>** definisce una didascalia per l'elemento **<fieldset>**.

```
<form action="/action_page.php">  
<fieldset>
```

```

<legend>Personal information:</legend>
First name:<br>
<input type="text" name="firstname" value="Mickey"><br>
Last name:<br>
<input type="text" name="lastname" value="Mouse"><br><br>
<input type="submit" value="Submit">
</fieldset>
</form>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_form\_legend

```

L'elemento <select> - https://www.w3schools.com/html/html_form_elements.asp

L' elemento <select> definisce un **elenco a discesa**.

```

<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
</select>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_elem\_select

```

Gli elementi <option> definiscono un'opzione che può essere selezionata.

Per impostazione predefinita, il primo elemento nell'elenco a discesa è selezionato.

Per definire un'opzione preselezionata, aggiungi l'attributo **selected** all'opzione.

```
<option value="fiat" selected>Fiat</option>
```

Valori visibili:

Utilizza l'attributo **size** per specificare il numero di valori visibili.

```

<select name="cars" size="2">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
</select>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_elem\_select\_size

```

Consenti selezioni multiple:

Utilizzare l'attributo **multiple** per consentire all'utente di selezionare più di un valore.

```

<select name="cars" size="3" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
</select>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_elem\_select\_multiple

```

L'elemento <textarea>

L'elemento <textarea> definisce un campo di input su più righe (un'area di testo).

```

<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_elem\_textarea

```

L'attributo **rows** specifica il numero visibile di linee in un'area di testo.

L'attributo **cols** specifica la larghezza visibile di un'area di testo.

Puoi anche definire la **dimensione** dell'area di testo usando i CSS

```
<textarea name="message" style="width:200px; height:600px;">
```

L'elemento <button>

L' elemento <button> definisce un **pulsante** cliccabile.

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_elem_button

L'elemento <datalist> - https://www.w3schools.com/tags/tag_datalist.asp

L'elemento <**datalist**> specifica un elenco di opzioni predefinite per un elemento <**input**>.

Gli utenti vedranno un elenco a discesa delle opzioni predefinite mentre inseriscono i dati.

L'attributo **list** dell'elemento <**input**>, deve fare riferimento all'attributo **id** dell'elemento <**datalist**>.

```
<form action="/action_page.php">
<input list="browsers">
<datalist id="browsers">
    <option value="Internet Explorer">
    <option value="Firefox">
    <option value="Chrome">
</datalist>
</form>
```

L'elemento <output>

L'elemento <**output**> rappresenta il risultato di un calcolo (come quello eseguito da uno script).

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_elem_output

Tipi di input - https://www.w3schools.com/html/html_form_input_types.asp

- <**input type="button"**>, pulsante
- <**input type="checkbox"**>, casella di controllo
- <**input type="color"**>, campi di input che dovrebbe contenere un colore
- <**input type="date"**>, campi di input che dovrebbe contenere una data
- <**input type="datetime-local"**>, specifica un campo di inserimento data e ora, senza fuso orario
- <**input type="email"**>, campi di input che dovrebbe contenere un indirizzo e-mail
- <**input type="file"**>, campo di file di selezione e un pulsante "Sfoglia" per il caricamento dei file
- <**input type="hidden"**>, campo di input nascosto non viene mostrato all'utente, ma i dati vengono inviati quando il modulo viene inviato.
- <**input type="image"**>, definisce un'immagine come pulsante di invio
- <**input type="month"**>, permette all'utente di selezionare un mese e anno
- <**input type="number"**>, definisce una numerico campo di immissione
- <**input type="password"**>, campo password
- <**input type="radio"**>, pulsante di opzione(selezionare SOLO UNO)
- <**input type="range"**>, definisce un controllo per immettere un numero il cui valore esatto non è importante (come un cursore). Range di default è 0 a 100, è possibile impostare restrizioni
- <**input type="reset"**>, pulsante di ripristino che ripristinerà tutti i valori del modulo ai valori predefiniti
- <**input type="search"**>, si usa per i campi di ricerca
- <**input type="submit"**>, invio dei dati del modulo a un gestore di moduli
- <**input type="tel"**>, campi di input che dovrebbe contenere un numero di telefono
- <**input type="text"**>, campo di immissione del testo su una riga
- <**input type="time"**>, permette all'utente di selezionare un tempo
- <**input type="url"**>, campi di input che dovrebbe contenere un indirizzo URL
- <**input type="week"**>, permette all'utente di selezionare una settimana e anno

Attributi di input - https://www.w3schools.com/html/html_form_attributes.asp

L'attributo **value** specifica il valore iniziale per un campo di input. <**form action=""**>

L'attributo *readonly*

L'attributo ***readonly*** specifica che il campo di input è di sola lettura (non può essere modificato).

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" readonly>
</form>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_input\_attributes\_READONLY
```

L'attributo *disabilitato*

L'attributo ***disabled*** specifica che il campo di input è disabilitato, è inutilizzabile e non selezionabile e il suo valore non verrà inviato quando si invia il modulo.

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" disabled>
</form>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_input\_attributes\_DISABLED
```

La dimensione Attributo

L'attributo ***size*** specifica la dimensione (in caratteri) per il campo di input.

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" size="40">
</form>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_input\_attributes\_SIZE
```

L'attributo ***maxlength*** specifica la lunghezza massima consentita per il campo di input

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" maxlength="10">
</form>          https://www.w3schools.com/html/tryit.asp?filename=tryhtml\_input\_attributes\_MAXLENGTH
```

L'attributo *segnaposto*

L'attributo ***placeholder*** specifica un suggerimento che descrive il valore atteso di un campo di input (un valore di esempio o una breve descrizione del formato)

```
<input type="text" name="fname" placeholder="First name">
https://www.w3schools.com/html/tryit.asp?filename=tryHTML5\_input\_placeholder
```

Elemento *<meta>* - https://www.w3schools.com/tags/tag_meta.asp

I ***metadati*** sono dati (informazioni) sui dati.

Il tag ***<meta>*** fornisce metadati relativi al documento HTML che non verranno visualizzati nella pagina, ma saranno analizzabili dalla macchina.

Gli ***elementi meta*** vengono in genere utilizzati per specificare la descrizione della pagina, le parole chiave, l'autore del documento, l'ultima modifica e altri metadati.

I ***metadati*** possono essere utilizzati dai browser (come visualizzare il contenuto o la pagina di ricarica), i motori di ricerca (parole chiave) o altri servizi Web.

Suggerimenti e note:

- i tag ***<meta>*** vanno sempre all'interno dell'elemento ***<head>***

- i **metadati** vengono sempre passati come *coppie nome / valore*.
- l'attributo **content** DEVE essere definito se il **name** o l'attributo **http-equiv** è definito.

Impostazione di Viewport

HTML5 ha introdotto un metodo per consentire ai web designer di assumere il controllo sulla vista, tramite il tag <meta>.

Il **viewport** è l'area visibile dell'utente di una pagina Web, varia a seconda del dispositivo e sarà più piccolo su un telefono cellulare che sullo schermo di un computer.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- Un elemento <meta> viewport fornisce al browser le istruzioni su come controllare le dimensioni e il ridimensionamento della pagina.
- La parte **width=device-width** imposta la larghezza della pagina in modo che segua la larghezza dello schermo del dispositivo (che varierà a seconda del dispositivo).
- La parte **initial-scale=1.0** imposta il livello di zoom iniziale quando la pagina viene caricata per la prima volta dal browser.

Differenze tra HTML 4.01 e HTML5

L'attributo dello schema non è supportato in HTML5, quest'ultimo ha un nuovo attributo, **charset**, che semplifica la definizione del charset:

HTML 4.01: <meta http-equiv = "tipo di contenuto" content = "text / html; charset = UTF-8">

HTML5: <meta charset = "UTF-8">

c:forEach - https://www.tutorialspoint.com/jsp/jstl_core.foreach_tag.htm

Questi tag esistono come una buona alternativa all'inclusione di un ciclo Java **for**, **while** o **do-while** tramite uno **scriptlet**. Il tag <c: forEach> è un tag comunemente usato perché itera su una collezione di oggetti. Il tag <c: forTokens> viene utilizzato per spezzare una stringa in token e scorrere tra i token.

Attributo:

Attribute	Description	Required	Default
items	Information to loop over	No	None
begin	Element to start with (0 = first item, 1 = second item, ...)	No	0
end	Element to end with (0 = first item, 1 = second item, ...)	No	Last element
step	Process every step items	No	1
var	Name of the variable to expose the current item	No	None
varStatus	Name of the variable to expose the loop status	No	None

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title><c:forEach> Tag Example</title>
  </head>

  <body>
    <c:forEach var = "i" begin = "1" end = "5">
      Item <c:out value = "${i}" /><p>
    </c:forEach>
  </body>
</html>
```

Il codice sopra genererà il seguente risultato:

```
Item 1
Item 2
Item 3
Item 4
Item 5
```

<c: out> - https://www.tutorialspoint.com/jsp/jstl_core_out_tag.htm

Il **tag <c: out>** mostra il risultato di un'espressione. Questo è quasi simile al modo in cui **<%=%>** funziona. La differenza qui è che il **tag <c: out>** ti permette di usare il più semplice "." notazione per accedere alle proprietà. Ad esempio, per accedere a customer.address.street, utilizzare il tag **<c: out value = "customer.address.street" />**.

Il **tag <c: out>** può sfuggire automaticamente ai tag XML in modo che non vengano valutati come tag effettivi.

Attribute	Description	Required	Default
Value	Information to output	Yes	None
default	Fallback information to output	No	body
escapeXml	True if the tag should escape special XML characters	No	true

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> <c:out> Tag Example</title>
  </head>

  <body>
    <c:out value = "${'<tag> , &'}" />
  </body>
</html>
```

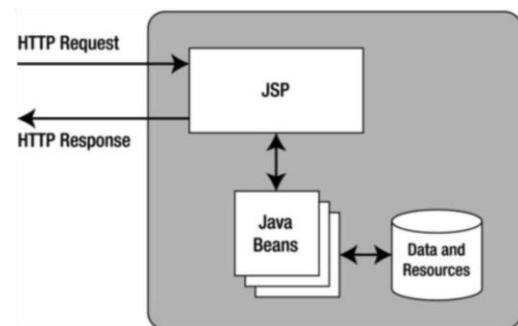
The above code will generate the following result –

```
<tag> , &
```

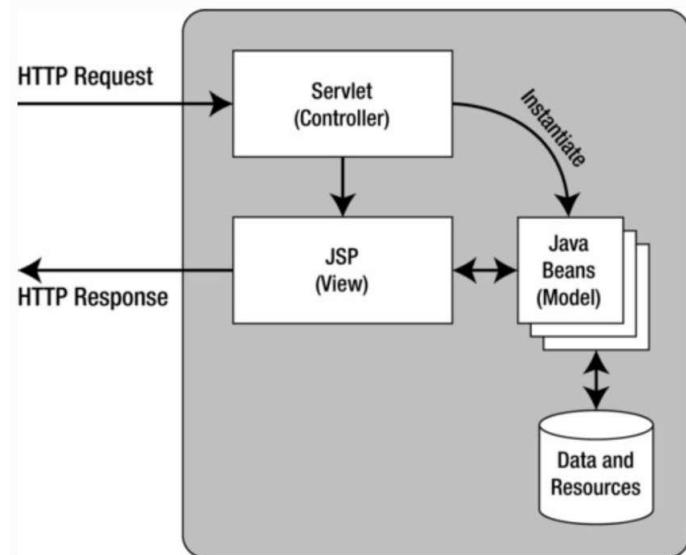
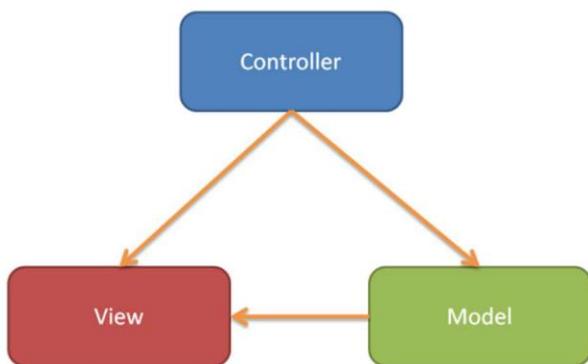

Modello MVC

Nel progetto di applicazioni Web in Java, due modelli di ampio uso e di riferimento:

- **Model 1** è un pattern semplice in cui codice responsabile per presentazione contenuti è mescolato con logica di business, Suggerito solo per piccole applicazioni (sta diventando obsoleto nella pratica industriale)



- **Model 2, Model-View-Controller (MVC)** come design pattern più complesso e articolato che separa chiaramente il livello **presentazione** dei contenuti dalla **logica** utilizzata per manipolare e processare i contenuti stessi



1. **Controller** – definisce il comportamento dell'applicazione
 - fa *dispatching* di richieste utente e seleziona la *view* per la presentazione
 - interpreta l'input dell'utente e lo mappa su azioni che devono essere eseguite da *model*
2. **Model** – rappresenta livello dei dati, incluse operazioni per accesso e modifica.
 - possibilità per la *view* di interrogare stato di *model*
 - possibilità per il *controller* di accedere alle funzionalità incapsulate dal *model*
3. **View** – si occupa di visualizzare i contenuti del *model*. Accede ai dati tramite il *model* e specifica come dati debbano essere presentati
 - aggiorna la presentazione dei dati quando il *model* cambia
 - Fornisce i moduli per l'invio dell'input dell'utente verso il *controller* da parte del browser

Esecuzione:

In applicazioni Web conformi al **Model 2**, richieste del browser client vengono passate al **Controller** (implementato da una **Servlet**).

Il **Controller** si occupa di eseguire la logica business necessaria per ottenere il contenuto da mostrare. Il **Controller** mette il contenuto nel **Model** (implementato con **JavaBean**) in un messaggio e decide a quale **View** (implementata da **JSP**) passare la richiesta.

La **View** si occupa del rendering del contenuto.

SERVLET

Servlet job:

- Legge dati esplicativi inviati dal client (Form data);
- Legge dati impliciti inviati dal client (Request Headers);
- Genera i risultati;
- Manda i dati esplicativi al client (HTML);
- Manda i dati impliciti al client (Status Code e Response Headers).

Una servlet è una classe java che si interfaccia con http.

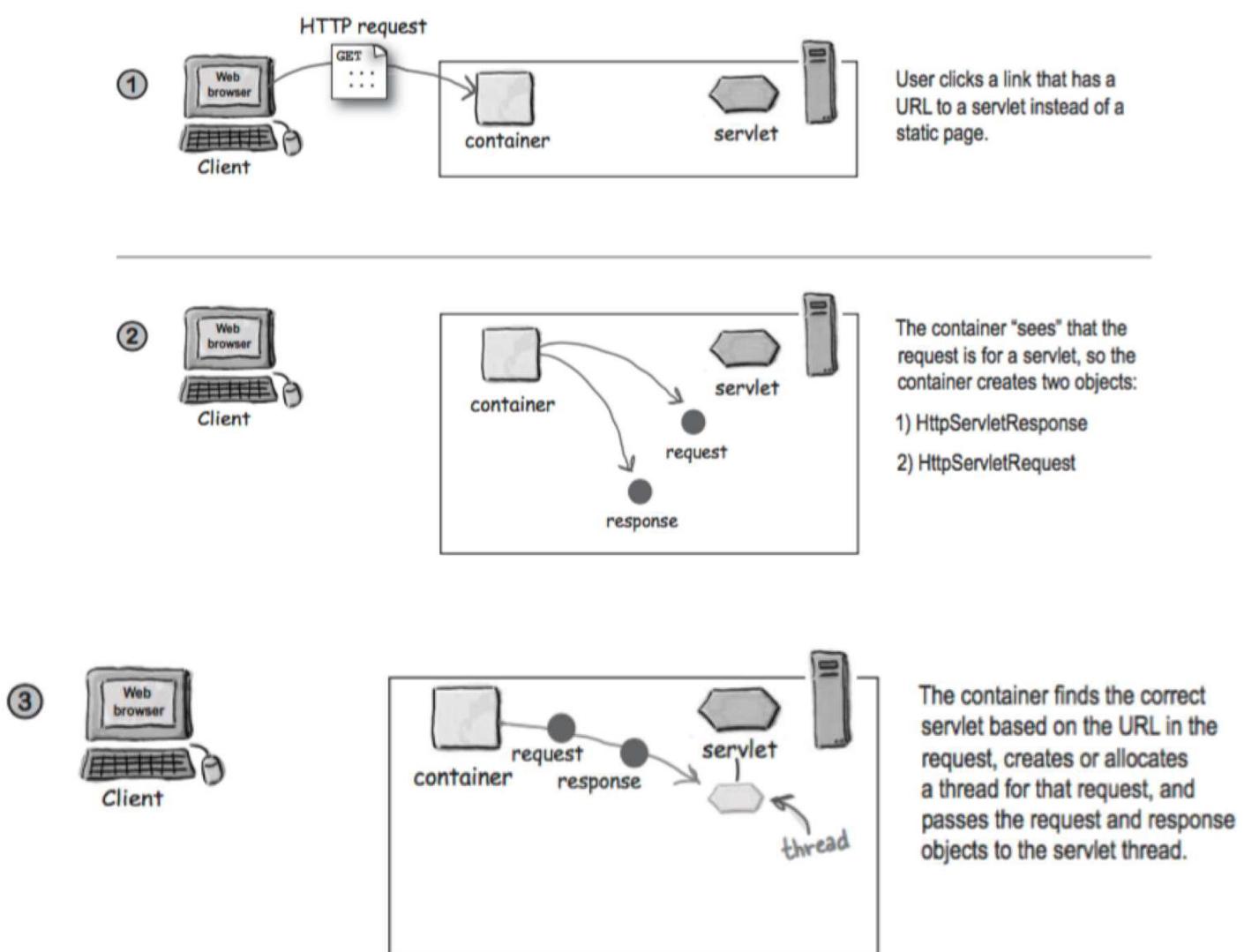
La Servlet crea **pagine dinamiche**, vengono presi informazioni da database o da altri server, essendo pagine che cambiano frequentemente.

Apache/Tomcat è definito compiutamente come Web/Servlet Container.

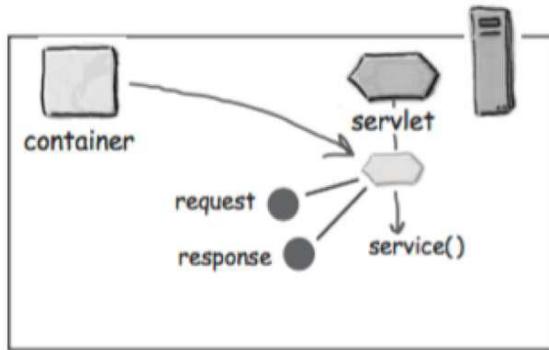
Le **Servlet** non hanno un “metodo main”, sono sotto il controllo di un’altra applicazione java chiamata **Container** (TomCat).

Quando una server web application come **Apache** manda una richiesta alla servlet, il server non manda la richiesta alla servlet stessa ma ad un **Container** all’interno del quale è contenuta la **Servlet**. Il **Container** fornisce alla servlet request e response, chiama i metodi della servlet come **doPost** e **doGet**, controllando vita e morte della Servlet.

Il **Container** automaticamente crea nuovi **Thread** java per ogni **Request** che riceve.



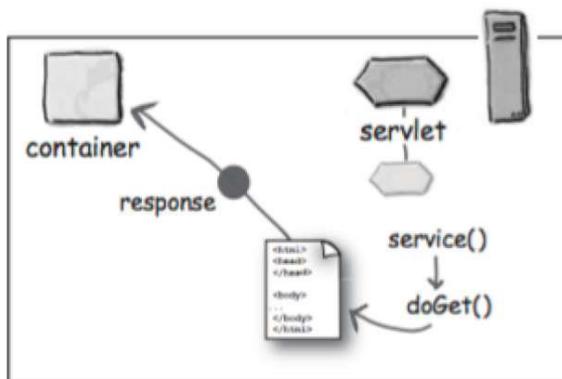
④



The container calls the servlet's `service()` method. Depending on the type of request, the `service()` method calls either the `doGet()` or `doPost()` method.

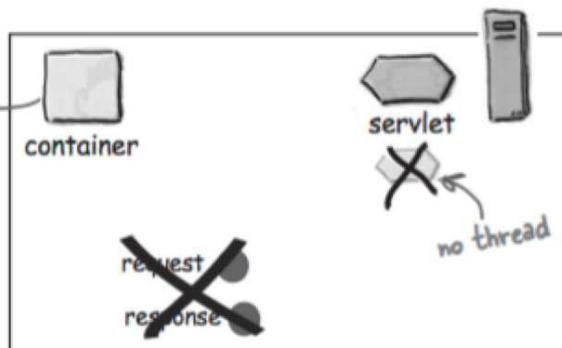
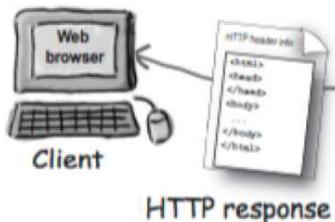
For this example, we'll assume the request was an HTTP GET.

⑤



The `doGet()` method generates the dynamic page and stuffs the page into the response object. Remember, the container still has a reference to the response object!

⑥



The thread completes, the container converts the response object into an HTTP response, sends it back to the client, then deletes the request and response objects.

La **Servlet** una classe Java che fornisce **risposte a richieste HTTP**. In termini più generali, è una classe che fornisce un servizio comunicando con il client mediante protocolli di tipo **request/response**, ad esempio **HTTP**.

Le **Servlet** estendono le funzionalità di un **Web Server** generando contenuti dinamici, che vengono eseguite direttamente in un **Web Container**.

In termini pratici sono classi che derivano dalla classe **HttpServlet**, che possono essere ridefinite, come il **doGet()**

```
...
public class HelloServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<title>Hello World!</title>");
    }
    ...
}
```

Le **Servlet** sono classi Java che elaborano richieste seguendo un protocollo condiviso.

Le **Servlet HTTP** sono il tipo più comune di Servlet e possono processare **richieste HTTP**, producendo **response http**.

Ogni Servlet, che comunicare coi client mediante protocollo http, deve derivare da: javax.servlet.http.HttpServlet.

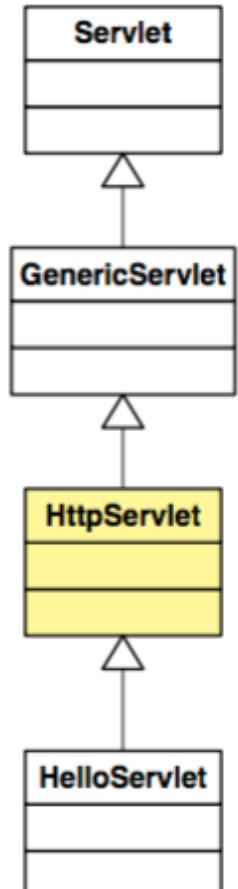
Un contatore del numero complessivo di utenti on-line di un sito Web, è opportuno collocarlo al livello application.

Diverse servlet di una applicazione web possono condividere degli oggetti mediante ServletContext.

Se ci sono più utenti su macchine diverse che accedono alla stessa servlet, lo scope di applicazione permette loro di condividere lo stesso dato.

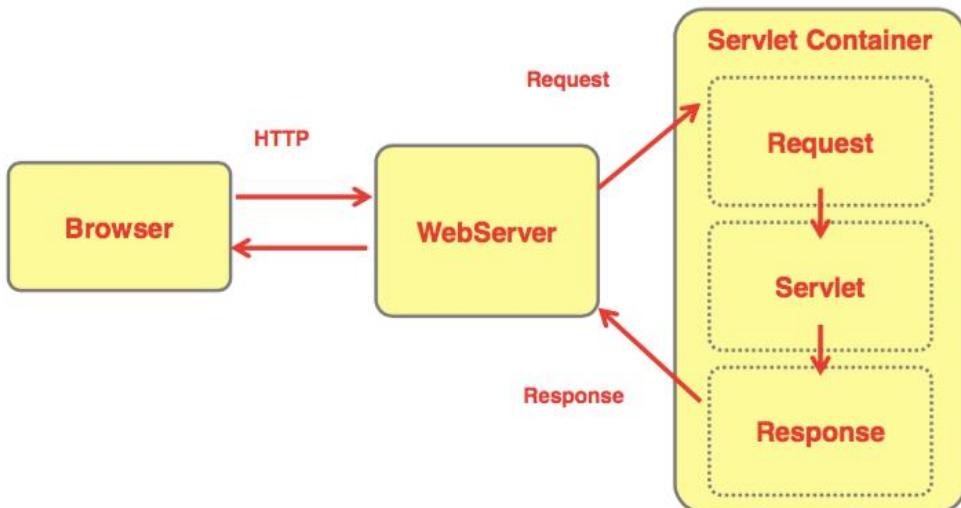
Una variabile di istanza di una servlet viene condivisa da tutti gli utenti di quella servlet.

Lo scope di request fa condividere uno stesso dato a più servlet o jsp a patto che queste stiano eseguendo all'interno della stessa request.



REQUEST / RESPONSE

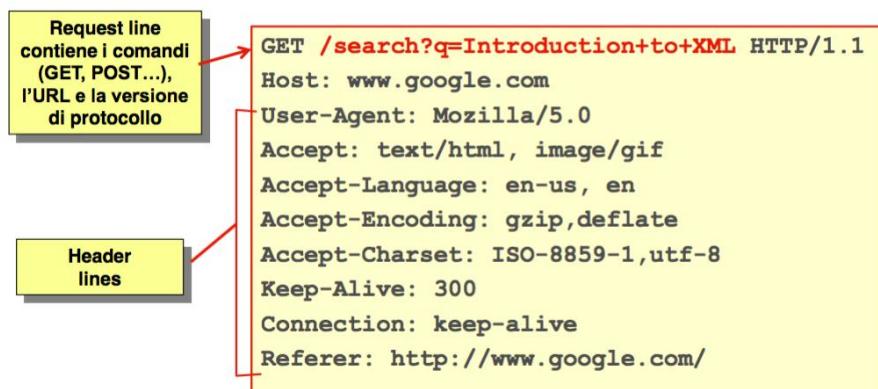
All'arrivo di una richiesta **HTTP** il **Servlet Container (Web Container)** crea un oggetto **request** e un oggetto **response** e li passa alla **Servlet**.



Gli oggetti di tipo **Request** rappresentano la chiamata al Server effettuata dal Client. Caratterizzati da varie informazioni.

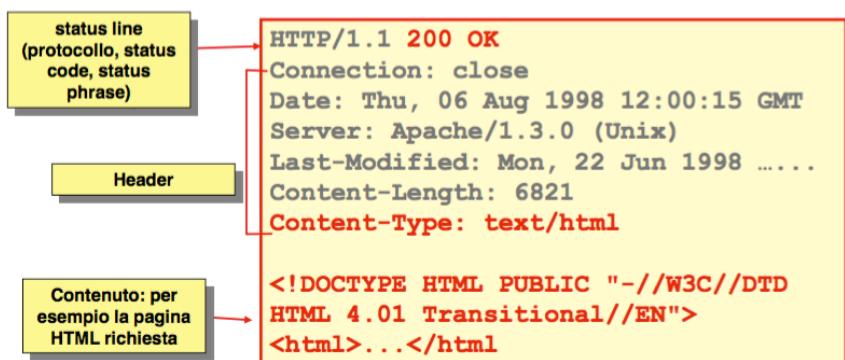
Viene creata dal **Servlet container** e passata alla Servlet come parametro ai metodi **doGet()** e **doPost()**. È un'istanza di una classe che implementa l'interfaccia **HttpServletRequest**:

- Chi ha effettuato la Request
- Quali parametri sono stati passati nella Request
- Quali header sono stati passati



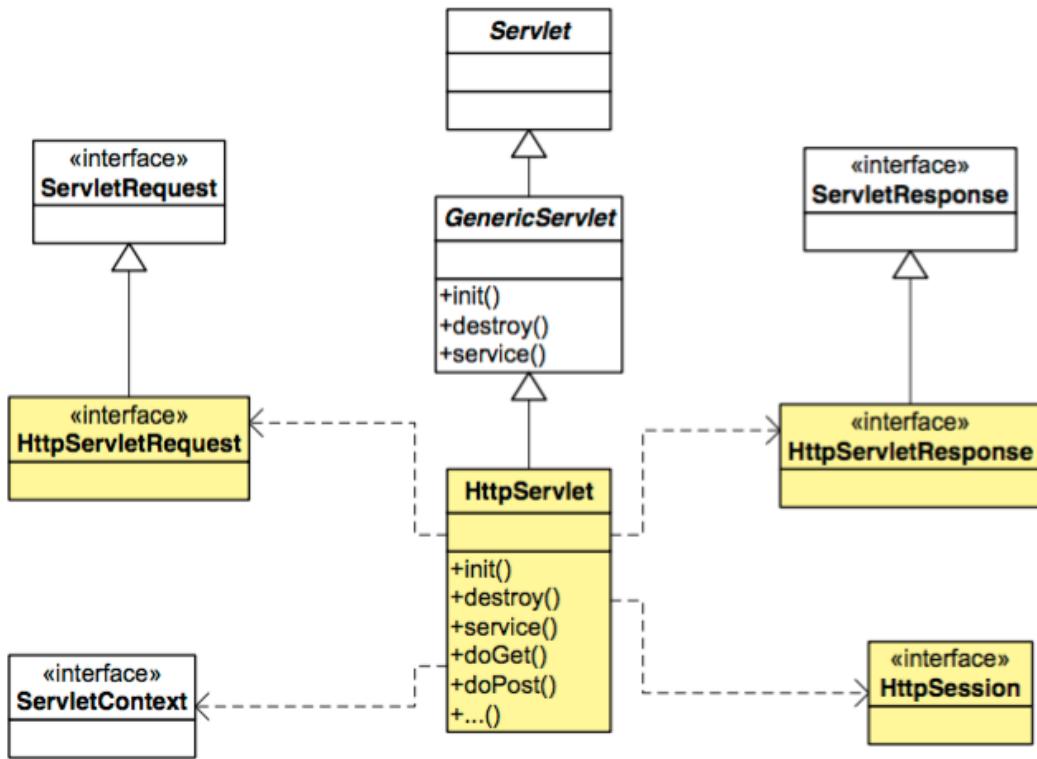
Gli oggetti di tipo **Response** rappresentano le informazioni restituite al client in risposta ad una Request:

- **Status line** (status code, status phrase)
- **Header** della risposta HTTP
- **Response body**: il contenuto (ad es. pagina HTML)



In rosso ciò che può/deve essere specificato a livello di codice della servlet

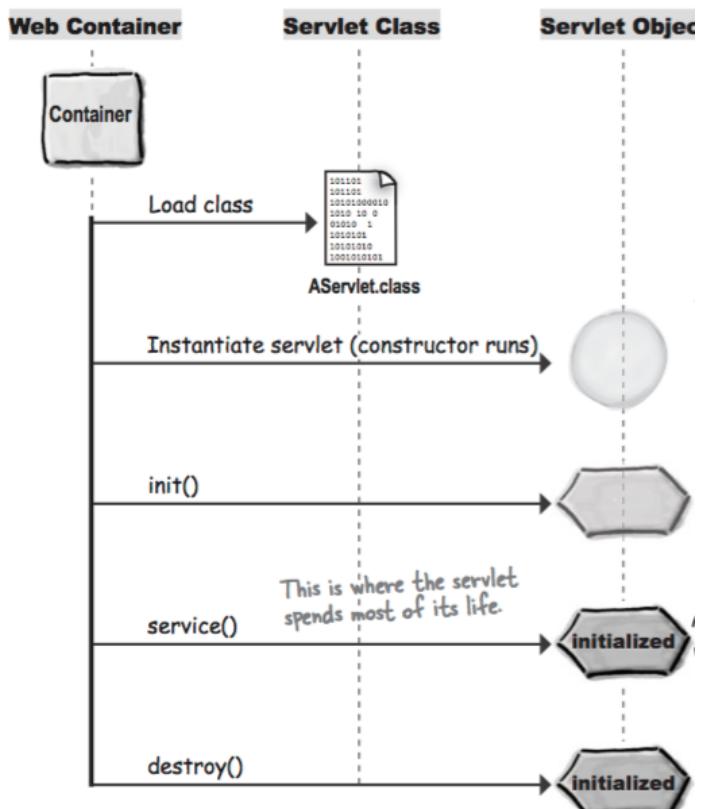
CLASSI E INTERFACCE PER SERVLET



- **HttpServlet** fornisce una implementazione di `service()` che delega l'elaborazione della richiesta ai metodi: `doGet()`, `doPost()`,

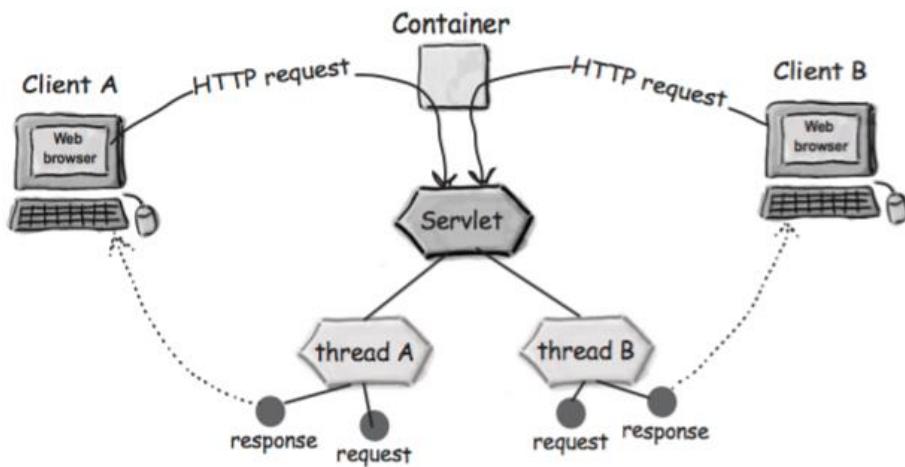
Ciclo di vita delle Servlet

1. **INIT()**: Il **Container** chiama la `init()` sull'istanza della **servlet** dopo che questa è stata creata, ma prima la **servlet** può servire altre richieste **client**. Se si vuole si può modificare il metodo(es. per connettersi al DB o ad altri oggetti). Può essere sovrascritta.
2. **SERVICE()**: Quando un **client** effettua la sua prima richiesta, il **container** crea un nuovo **thread** invocando il metodo `service()`, che guarda la richiesta(GET, POST, ecc..) ed invoca la `doGet()` e la `doPost()`. Il `service()` non deve essere mai sovrascritta.
3. **DESTROY()**: viene chiamato una sola volta quando la Servlet deve essere disattivata e serve per rilasciare le risorse acquisite.



Modello “normale”: una sola istanza di Servlet e un thread assegnato ad ogni richiesta http per Servlet, anche se richieste per quella Servlet sono già in esecuzione. Nella modalità normale più thread condividono la stessa istanza di una Servlet e quindi si crea una situazione di concorrenza.

MULTITHREADING: Il *container* esegue più *thread* di più *processi* richiesti da una singola servlet:



DEPLOYMENT

Un'applicazione Web deve essere installata e questo processo prende il nome di **deployment** che comprende:

- La definizione del **runtime environment** (ambiente di esecuzione) di una Web Application
- La mappatura delle **URL** sulle Servlet
- La definizione delle impostazioni di **default** di un'applicazione
- La configurazione delle caratteristiche di **sicurezza** dell'applicazione

Web.xml È un file di configurazione (in formato XML) che contiene una serie di elementi descrittivi, descrive la struttura dell'applicazione Web.

Contiene l'elenco delle Servlet e per ognuna di loro permette di definire una serie di parametri come coppie nome-valore:

- Nome
- Classe Java corrispondente
- Una serie di parametri di configurazione (coppie nome-valore, valori di inizializzazione)
- Contiene mappatura fra URL e Servlet che compongono l'applicazione (**IMPORTANTE!**)

Mappatura Servlet-URL:

Descrittore con mappatura:

```
<web-app>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>myPackage.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myURL</url-pattern>
  </servlet-mapping>
</web-app>
```

URL che viene mappato su **myServlet**:

<http://MyHost:8080/MyWebApplication/myURL>

Servlet configuration:

Una Servlet accede ai propri parametri di configurazione mediante l'interfaccia **ServletConfig**. Ci sono 2 modi per accedere a oggetti di questo tipo:

1. Il parametro di tipo **ServletConfig** passato al metodo **init()**
2. il metodo **getServletConfig()** della Servlet, che può essere invocato in qualunque momento

ServletConfig espone un metodo per ottenere il valore di un parametro in base al nome:

- **String getInitParameter(String parName):**

Esempio di parametro
di configurazione

```
<init-param>
    <param-name>parName</param-name>
    <param-value>parValue</param-value>
</init-param>
```

Pagine di errore:

```
<error-page>
    <error-code>404</error-code>
    <location>/commons/redirectToError.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/commons/fatalError.jsp</location>
</error-page>
```

Pagine di eccezione:

```
<error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/error.html</location>
</error-page>
```

Servlet context

Un **Servlet Context** è costituito da un gruppo di Servlet, pagine JSP o altre pagine web che condividono tra di loro risorse e dati. Ogni **Servlet Context** corrisponde ad una applicazione web, e viene associato ad un unico path di prefisso denominato context path.

Si può ottenere un'istanza di tipo **ServletContext** all'interno della Servlet utilizzando il metodo **getServletContext()** consentendo l'accesso ai parametri di inizializzazione e alle risorse statiche della Web Application mediante il metodo **InputStream getResourceAsStream(String path)**.

IMPORTANTE: Servlet context viene condiviso tra tutti gli utenti, le richieste e le Servlet facenti parte della stessa Web application.

Parametri di inizializzazione del contesto definiti all'interno di elementi di tipo **context-param** in **web.xml**:

```
<web-app>
    <context-param>
        <param-name>feedback</param-name>
        <param-value>feedback@deis.unibo.it</param-value>
    </context-param>
    ...
</ web-app >
```

Sono accessibili a tutte le **Servlet** della **Web application**:

```
...
ServletContext ctx = getServletContext();
String feedback =
ctx.getInitParameter("feedback");
...
```

Gli **attributi di contesto** sono accessibili a tutte le **Servlet** e funzionano come **variabili "globali"**.

Vengono gestiti a runtime e possono essere creati, scritti e letti dalle Servlet.

scrittura

```
ServletContext ctx = getServletContext();
ctx.setAttribute("utente1", new User("Giorgio Bianchi"));
ctx.setAttribute("utente2", new User("Paolo Rossi"));
```

lettura

```
ServletContext ctx = getServletContext();
Enumeration aNames = ctx.getAttributeNames();
while (aNames.hasMoreElements)
{
    String aName = (String)aNames.nextElement();
    User user = (User) ctx.getAttribute(aName);
    ctx.removeAttribute(aName);
}
```

Gestione dello stato (di sessione)

HTTP è un protocollo stateless: non fornisce in modo nativo meccanismi per il mantenimento dello stato tra diverse richieste provenienti dallo stesso **client**.

Le applicazioni Web hanno spesso bisogno di stato. Sono state definite due tecniche per mantenere traccia delle informazioni di stato:

- uso dei cookie: meccanismo di basso livello
- uso della sessione (session tracking): meccanismo di alto livello

La sessione rappresenta un'utile astrazione ed essa stessa può far ricorso a tre meccanismi base di implementazione:

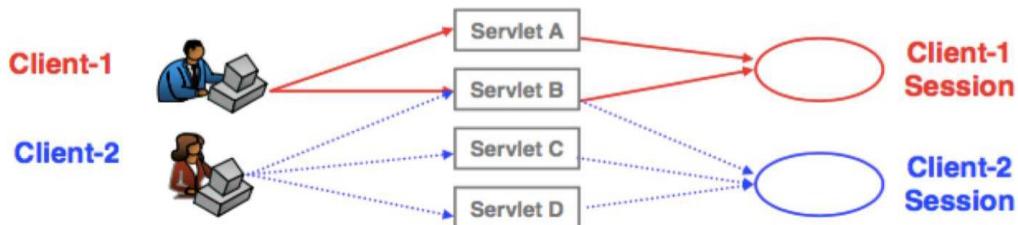
1. Cookie
2. URL rewriting
3. Hidden form fields

SESSIONE

La sessione Web è un'entità gestita dal **Web Container**, ed è condivisa fra tutte le richieste provenienti dallo stesso client: consente di mantenere, quindi, informazioni di stato (di sessione).

Può contenere dati di varia natura ed è identificata in modo univoco da una **session ID**.

Questo è creato dall'applicazione (servlets) e dovrà essere sempre presente in ogni richiesta che fa l'utente, affinché l'utente possa essere riconosciuto dalle servlets.



Come faccio io programmatore Web a fare in modo che il client mi invii il session Id che io gli ho assegnato ogni qualvolta che mi manda una richiesta se i cookie sono disattivati?

1. URL-Rewriting

Per ogni **URL** che il server restituisce al client, aggiunge alcuni dati aggiuntivi alla fine di ciascun URL che identifica la sessione. Il server associa quell'identificatore con i dati che ha archiviato su quella sessione. *Esempio: http://host/path/file.html; jsessionid = 1234*

Vantaggio:

- Funziona anche se i cookie sono disabilitati o non supportati

Svantaggi:

- Il programmatore web deve codificare tutti gli URL che si riferiscono al tuo sito
- Tutte le pagine devono essere generate dinamicamente
- Non riesce a trovare segnalibri e collegamenti da altri siti

2. Hidden form fields

`<input type="hidden" name="session" value="...">`

Vantaggio:

- Funziona anche se i cookie sono disabilitati o non supportati

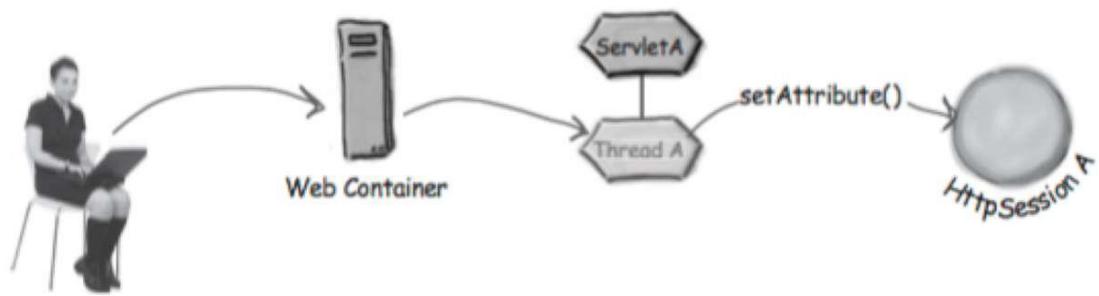
Svantaggi:

- Un sacco di elaborazione noiosa
- Tutte le pagine devono essere il risultato di invii di moduli

- ① Diane selects "Dark" and hits the submit button.

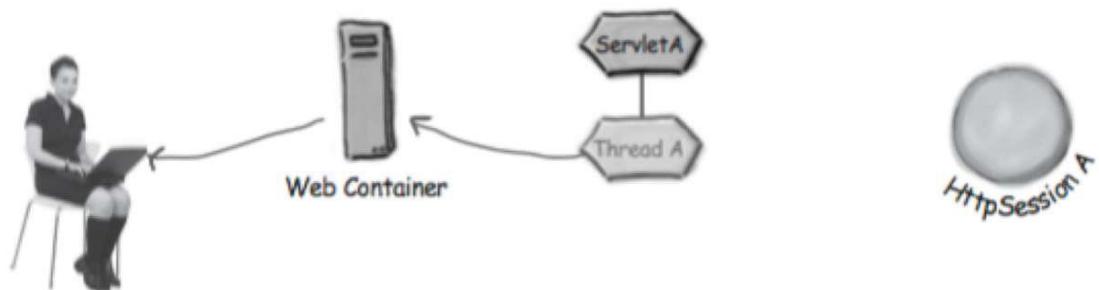
The Container sends the request to a new thread of the BeerApp servlet.

The BeerApp thread finds the session associated with Diane, and stores her choice ("Dark") in the session as an attribute.



- ②

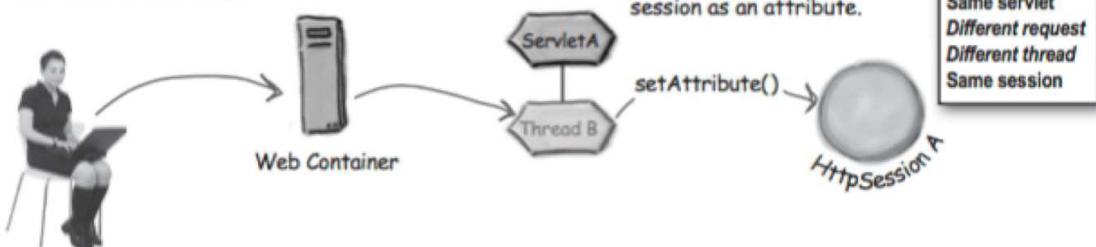
The servlet runs its business logic (including calls to the model) and returns a response... in this case another question, "What price range?"



- ③ Diane considers the new question on the page, selects "Expensive" and hits the submit button.

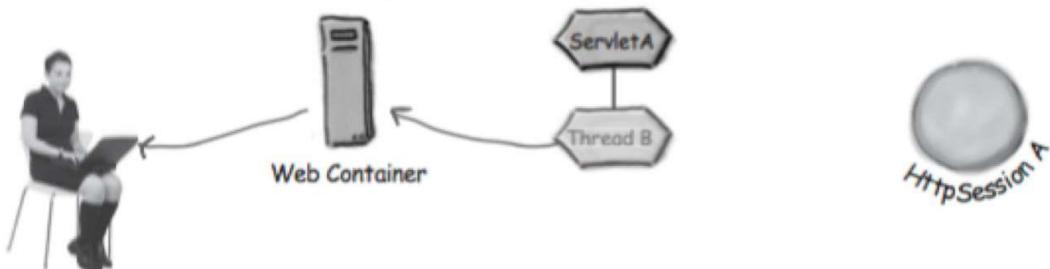
The Container sends the request to a new thread of the BeerApp servlet.

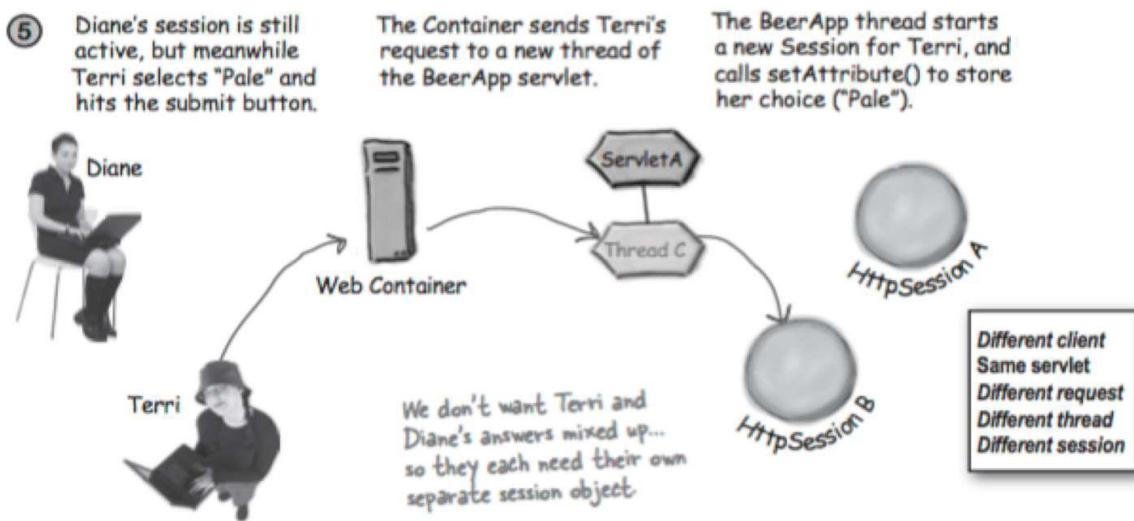
The BeerApp thread finds the session associated with Diane, and stores her new choice ("Expensive") in the session as an attribute.



- ④

The servlet runs its business logic (including calls to the model) and returns a response... in this case another question.





Accesso alla sessione

L'accesso avviene mediante l'interfaccia **`HttpSession`**. Per ottenere un riferimento ad un oggetto di tipo **`HttpSession`** si usa il metodo **`getSession()`** dell'interfaccia **`HttpServletRequest`**.

```
public HttpSession getSession(boolean createNew);
```

Valori di **`createNew`**:

- **true**: ritorna la sessione esistente o, se non esiste, ne crea una nuova
- **false**: ritorna, se possibile, la sessione esistente, altrimenti ritorna null

```
HttpSession ssn = request.getSession();
if(ssn != null){
    String ssnId = ssn.getId();
    System.out.println("Your session Id is : "+ ssnId);
}
```

Si possono memorizzare ***dati specifici dell'utente negli attributi della sessione*** (coppie nome/valore). Sono simili agli attributi di contesto, ma con **scope(visibilità)** fortemente diverso, e consentono di memorizzare e recuperare oggetti.

- **`void setAttribute(String name, Object o);`**
- **`Object getAttribute(String name);`**
- **`void removeAttribute(String name);`**
- **`Enumeration.getAttributeNames();`**

```
Cart sc = (Cart) session.getAttribute("shoppingCart");
sc.addItem(item);
...
session.removeAttribute("shoppingCart");
...
session.setAttribute("shoppingCart", new Cart());
...
Enumeration e = session.getAttributeNames();
while(e.hasMoreElements())
    out.println("Key: " + (String)e.nextElement());
```

- **`String getId()`** restituisce l'ID di una sessione
- **`boolean isNew()`** dice se la sessione è nuova
- **`void invalidate()`** permette di invalidare (distruggere) una sessione

- **long getCreationTime()** dice da quanto tempo è attiva la sessione (in millisecondi)
- **long getLastAccessedTime()** dà informazioni su quando è stata utilizzata l'ultima volta

```
String sessionID = session.getId();
if(session.isNew())
    out.println("La sessione e' nuova");
session.invalidate();
out.println("Millisec:" + session.getCreationTime());
out.println(session.getLastAccessedTime());
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession();
    synchronized (session) {
        String heading;
        Integer accessCount = (Integer) session.getAttribute("accessCount");
        if (accessCount == null) {
            accessCount = 0;
            heading = "Welcome, Newcomer";
        } else {
            heading = "Welcome Back";
            accessCount = accessCount + 1;
        }
        session.setAttribute("accessCount", accessCount);

        PrintWriter out = response.getWriter();
        out.println (
            "<!DOCTYPE html>" +
            "<html>" +
            "<head><title>Session Tracking Example</title></head>" +
            "<body>" +
            "<h1>" + heading + "</h1>" +
            "<h2>Information on Your Session:</h2>" +
            "<table border='1'>" +
            "<tr>" +
            "  <th>Info Type</th><th>Value</th>" +
            "</tr><tr>" +
            "  <td>ID</td><td>" + session.getId() + "</td>" +
            "</tr><tr>" +
            "  <td>Creation Time</td><td>" + new Date(session.getCreationTime()) + "</td>" +
            "</tr><tr>" +
            "  <td>Time of Last Access</td><td>" + new Date(session.getLastAccessedTime()) + "</td>" +
            "</tr><tr>" +
            "  <td>Number of Previous Accesses</td><td>" + accessCount + "</td>" +
            "</table>" +
            "</body></html>");
    }
}
```

scope DIFFERENZIATI (scoped objects)

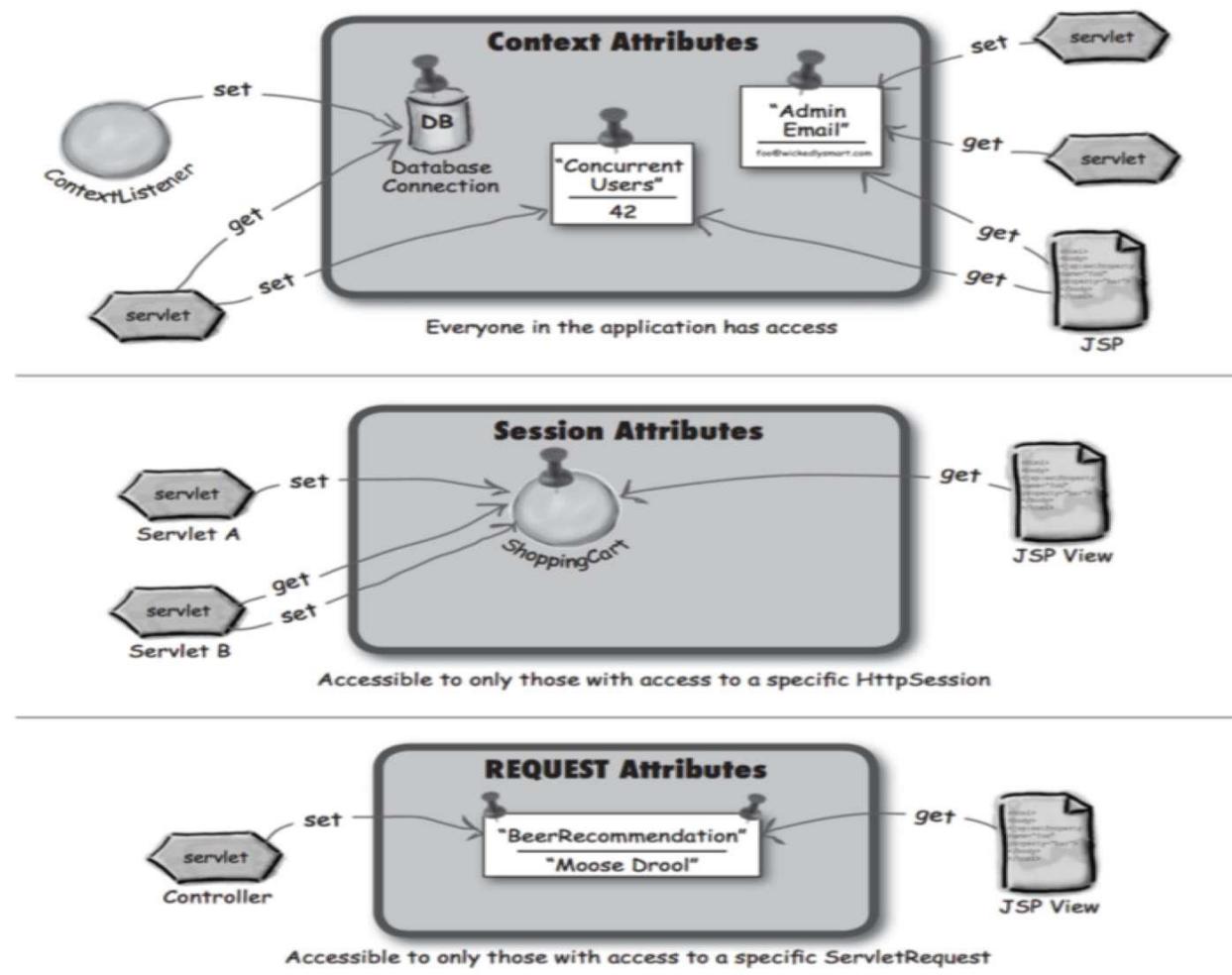
Gli oggetti di tipo **ServletContext**, **HttpSession**, **HttpServletRequest** forniscono metodi per immagazzinare e ritrovare oggetti nei loro rispettivi ambiti (*scope*):

- **void setAttribute(String name, Object o);**
- **Object getAttribute(String name);**
- **void removeAttribute(String name);**
- **Enumeration.getAttributeNames();**

Lo **scope** è definito dal tempo di vita (*lifespan*) e dall'accessibilità da parte delle Servlet.

Ambito	Interfaccia	Tempo di vita	Accessibilità
Request	HttpServletRequest	Fino all'invio della risposta	Servlet corrente e ogni altra pagina inclusa o in forward
Session	HttpSession	Lo stesso della sessione utente	Ogni richiesta dello stesso client
Application	ServletContext	Lo stesso dell'applicazione	Ogni richiesta alla stessa Web app anche da clienti diversi e per servlet diverse

The Three Scopes: Context, Request, and Session



Ridirezione del browser

È possibile inviare al browser una risposta che lo forza ad accedere ad un'altra pagina/risorsa (**ridirezione**)

```
String name=request.getParameter("name");
```

```
response.sendRedirect("https://www.google.co.in/#q="+name);
```

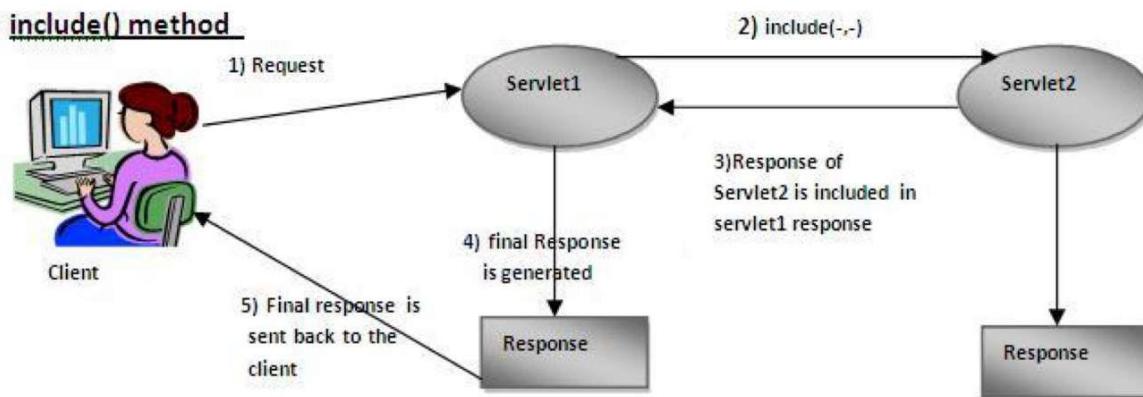
La ridirezione da parte di una servlet coinvolge il browser e cancella tutti dati nello scope di request

Includere una risorsa - include

Per includere una risorsa si ricorre a un oggetto di tipo **RequestDispatcher** che può essere richiesto al contesto indicando la risorsa da includere.

Si invoca quindi il metodo **include** passando come parametri **request** e **response** che vengono così condivisi con la risorsa inclusa. Se necessario, l'URL originale può essere salvato come un attributo di request.

```
RequestDispatcher dispatcher =  
    getServletContext().getRequestDispatcher("/inServlet");  
dispatcher.include(request, response);
```



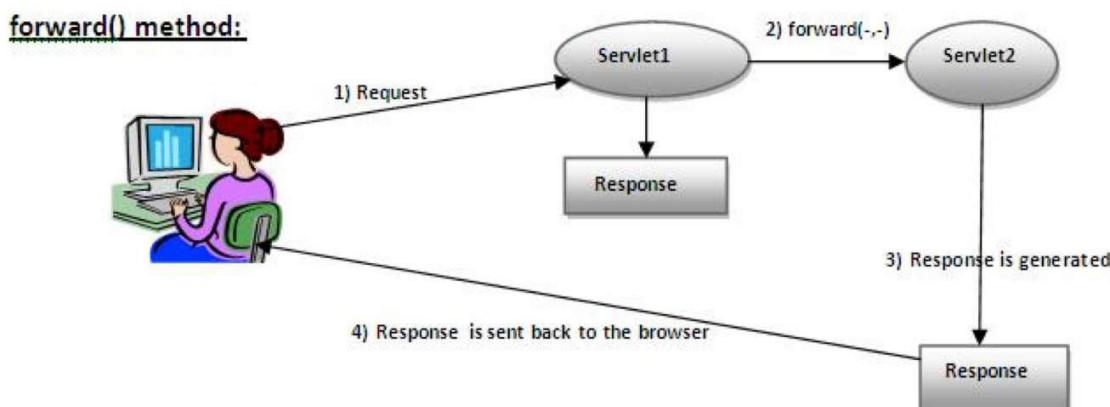
Inoltrare una risorsa - forward

Si usa in situazioni in cui una Servlet si occupa di parte dell'elaborazione della richiesta e delega a qualcun altro la gestione della risposta. *L'operazione di forward da parte di una servlet è invisibile al browser e mantiene i dati nello scope di sessione.*

Attenzione: in questo caso la risposta è di competenza esclusiva della risorsa che riceve l'inoltro

- Se nella prima **Servlet** è stato fatto un accesso a **ServletOutputStream** o **PrintWriter** si ottiene una **IllegalStateException**.
- Si deve ottenere un oggetto di tipo **RequestDispatcher** da request passando come parametro il nome della risorsa.
- Si invoca quindi il metodo **forward** passando anche in questo caso **request** e **response**.

```
RequestDispatcher dispatcher =  
    getServletContext().getRequestDispatcher("/inServlet");  
dispatcher.forward(request, response);
```



JSP: JAVA SERVER PAGES

Le JSP sono uno dei due componenti di base della tecnologia J2EE, relativamente alla parte Web:

- Template(programmi da riempire) per la generazione di contenuto dinamico
- estendono HTML con codice Java custom

Quando viene effettuata una richiesta a una JSP:

- parte dell'HTML viene direttamente trascritta sullo stream di output
- **Il codice Java viene eseguito sul server** per la generazione del contenuto HTML dinamico
- la pagina HTML così formata (parte statica + parte generata dinamicamente) viene restituita al client

Vengono trasformate in Servlet dal container

```
<html>
  <body>
    <% String visitor=request.getParameter("name");
       if (visitor == null) visitor = "World"; %>
    Hello, <%= visitor %>!
  </body>
</html>
```

<http://myHost/myWebApp/helloWord.jsp>

```
<html>
  <body>
    Hello, World!
  </body>
</html>
```

<http://myHost/myWebApp/helloWord.jsp?name=Mario>

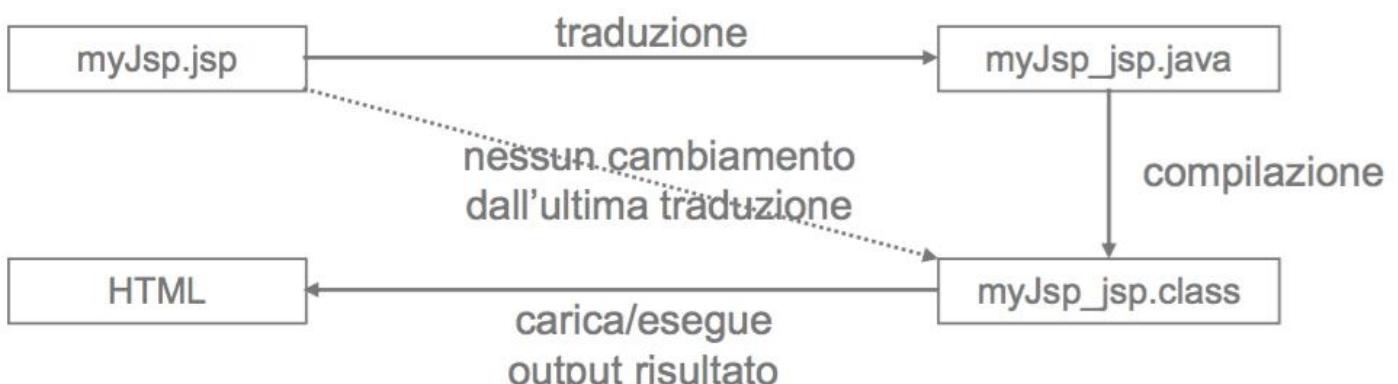
```
<html>
  <body>
    Hello, Mario!
  </body>
</html>
```

JspServlet

Le richieste verso JSP sono gestite da una particolare Servlet (in Tomcat si chiama **JspServlet**) che effettua:

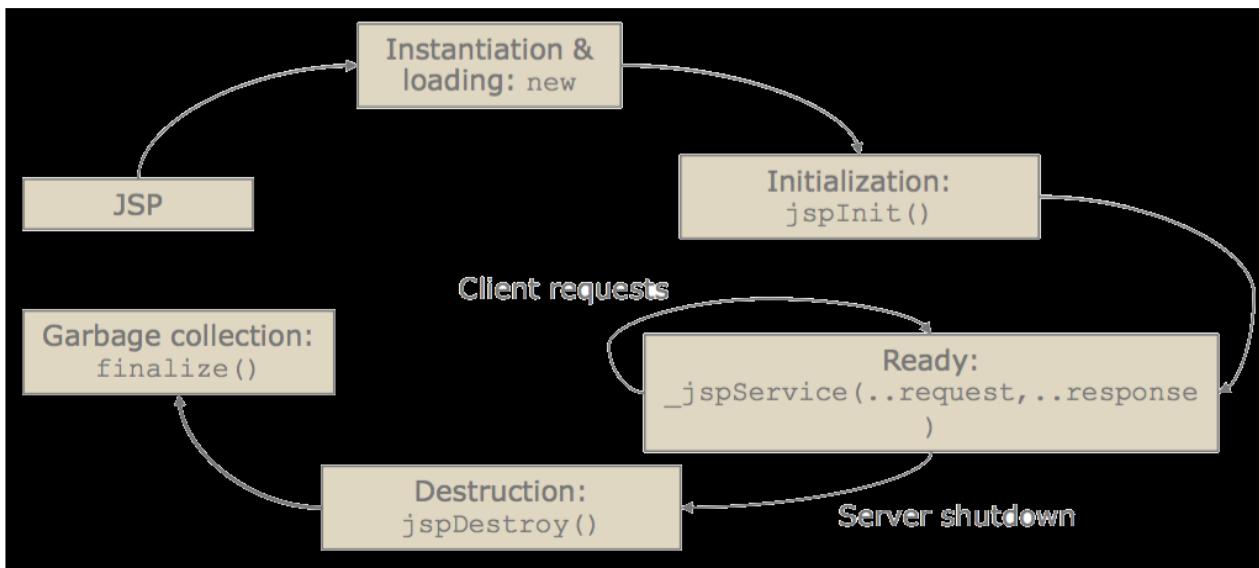
- traduzione della JSP in una Servlet
- compilazione della Servlet risultante in una classe
- esecuzione della JSP

I primi due passi vengono eseguiti solo quando cambia il codice della JSP



Ciclo di vita delle JSP

Dal momento che le **JSP** sono compilate in **Servlet**, il ciclo di vita delle JSP (*dopo la compilazione*) è controllato sempre dal medesimo **Web container**: `jsplInit()`, `_jspService()`, `jspDestroy()`.



Servlet vs JSP

Nella Servlet la logica per la generazione del documento HTML è implementata completamente in Java:

- Il processo di generazione delle pagine è *time-consuming*, ripetitivo e soggetto a errori (`println()`)
- L'aggiornamento delle pagine è scomodo

JSP nascono per facilitare la progettazione grafica e l'aggiornamento delle pagine:

- Si può separare agevolmente il lavoro fra grafici e programmatore
- I Web designer possono produrre pagine senza dover conoscere i dettagli della logica server-side
- La generazione di codice dinamico è implementata sfruttando il linguaggio Java

Tag delle JSP

Le parti variabili della pagina sono contenute all'interno di tag speciali. Ci sono 2 tipi di sintassi per i Tag:

- **XML-Oriented tag**
- **Scripting-oriented tag**, sono definite da delimitatori entro cui è presente lo scripting (self-contained), di 4 tipi:
 - **Dichiarazione**, si usano i delimitatori `<%!` e `%>` per dichiarare variabili e metodi, possono poi essere referenziati in qualsiasi punto del codice JSP. I metodi diventano metodi della Servlet quando la pagina viene tradotta

```
<%! String name = "Paolo Rossi";
   double[] prices = {1.5, 76.8, 21.5};

   double getTotal() {
       double total = 0.0;
       for (int i=0; i<prices.length; i++)
           total += prices[i];
       return total;
   }
%>
```

- **Espressioni**, si usano i delimitatori `<%= e %>` per valutare espressioni Java. Il risultato dell'espressione viene convertito in stringa inserito nella pagina al posto del tag

JSP

```
<p>Sig. <%=name%>, </p>
<p>l'ammontare del suo acquisto è: <%=getTotal()%> euro.</p>
<p>La data di oggi è: <%=new Date()%></p>
```



Pagina HTML risultante

```
<p>Sig. Paolo Rossi, </p>
<p>l'ammontare del suo acquisto è: 99.8 euro.</p>
<p>La data di oggi è: Tue Feb 20 11:23:02 2010</p>
```

- **Scriptlet**, si usano `<% e %>` per aggiungere un frammento di codice Java eseguibile alla JSP (**scriptlet**). Lo **scriptlet** consente di inserire logiche di controllo di flusso nella produzione della pagina.

```
<% if (userIsLogged) { %>
    <h1>Benvenuto Sig. <%=name%></h1>
<% } else { %>
    <h1>Per accedere al sito devi fare il login</h1>
<% } %>
```

- **Direttive**, sono comandi JSP valutati a tempo di compilazione. Le più importanti sono:

1. **page**: permette di importare package, dichiarare pagine d'errore, definire modello di esecuzione JSP relativamente alla concorrenza.
2. **include**: include un altro documento.
3. **taglib**: carica una libreria di custom tag implementate dallo sviluppatore

Non producono nessun output visible

```
<%@ page info="Esempio di direttive" %>
<%@ page language="java" import="java.net.*" %>
<%@ page import="java.util.List, java.util.ArrayList" %>
<%@ include file="myHeaderFile.html" %>
```

La direttiva page

```
<%@ page
[ language="java" ]
[ extends="package.class" ]
[ import="{package.class | package.*}, ..." ]
[ session="true | false" ]
[ buffer="none | 8kb | sizekb" ]
[ autoFlush="true | false" ]
[ isThreadSafe="true | false" ]
[ info="text" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType [ ;charset=characterSet ]"
  "text/html ; charset=ISO-8859-1" ]
[ isErrorPage="true | false" ]
%>
```

N.B. valori sottolineati sono quelli di default

- **language="java"** linguaggio di scripting utilizzato nelle parti dinamiche, allo stato attuale l'unico valore ammesso è "java"

- **import="package.class|package.*|..."** lista di package da importare.

Gli import più comuni sono impliciti e non serve inserirli (java.lang.*, javax.servlet.*, javax.servlet.jsp.*, javax.servlet.http.*)

- **session="true|false"** indica se la pagina fa uso della sessione (altrimenti non si può usare session)

- **buffer="none|8kb|sizekb"** dimensione in KB del buffer di uscita

- **autoFlush="true|false"** dice se il buffer viene svuotato automaticamente quando è pieno

Se il valore è false viene generata un'eccezione quando il buffer è pieno

- **isThreadSafe="true|false"** indica se il codice contenuto nella pagina è thread-safe

Se vale false le chiamate alla JSP vengono serializzate

- **info="text"** testo di commento

Può essere letto con il metodo Servlet.getServletInfo()

Esempio: <%= this.getServletInfo()%> oppure <%= page.getServletInfo()%>

- **errorPage="relativeURL"** indirizzo della pagina a cui vengono inviate le eccezioni

- **isErrorPage="true|false"** indica se JSP corrente è una pagina di errore

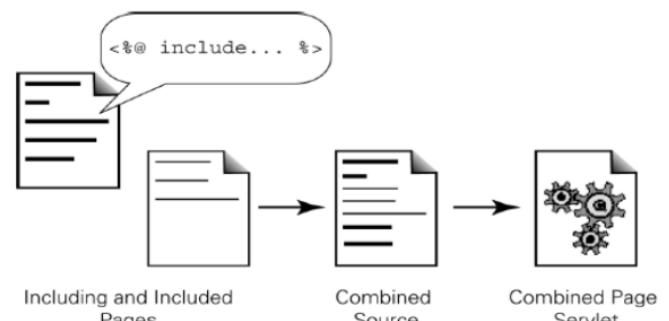
Si può utilizzare l'oggetto exception solo se l'attributo è true

- **contentType="mimeType [;charset=charSet]" | "text/html;charset=ISO- 8859-1"** indica il tipo MIME e il codice di caratteri usato nella risposta

La direttiva include

Serve ad includere file esterni durante la fase di conversione, nella servlet equivalente.

L'inclusione viene fatta a tempo di compilazione: eventuali modifiche al file incluso non determinano una ricompilazione della pagina che lo include.



```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>JSP Include</title>
</head>
<body>
<%@ include file="header.jsp" %>
<br>
Contact Us at: we@studytonight.com<br>
<br>
<%@ include file="footer.jsp" %>
</body>
</html>

```

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<div>
HEADER
</div>

```

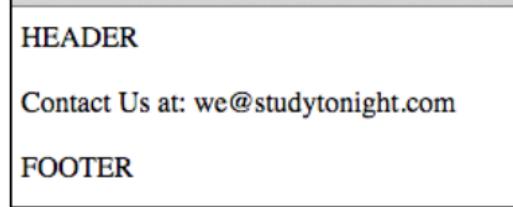
```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<div>
FOOTER
</div>

```

header.jsp

footer.jsp



Built-in objects (con scope differenziati)

Le specifiche JSP definiscono **8 oggetti built-in** (o impliciti) utilizzabili senza doverle definire ed istanziarle, le più usate sono: *page ,request ,session, application*.

Oggetto	Classe/Interfaccia
page	javax.servlet.jsp.HttpJspPage
config	javax.servlet.ServletConfig
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
pageContext	javax.servlet.jsp.PageContext
exception	Java.lang.Throwable

Oggetti impliciti di JSP - <https://www.w3adda.com/jsp-tutorial/jsp-implicit-objects>

Oggetto page

L'oggetto page rappresenta l'istanza corrente della Servlet. È un riferimento alla servlet, ovvero una istanza della servlet della pagina JSP che elabora la richiesta corrente. Può essere usato con *this*, usata per chiamare alcuni metodi definiti dalla classe servlet tradotta.

Ha come tipo l'interfaccia **HTTPJspPage** che discende da JSP page, la quale a sua volta estende Servlet. Può quindi essere utilizzato per accedere a tutti i metodi definiti nelle Servlet

```
<%@ page info="Esempio di uso page." %>
<p>Page info:
    <%=page.getServletInfo() %>
</p>
```

Pagina HTML

```
<p>Page info: Esempio di uso di page</p>
```

Oggetto config

Contiene la configurazione della Servlet (parametri di inizializzazione). /////Poco usato

Metodi di config:

- **getInitParameterNames()** restituisce tutti i nomi dei parametri di inizializzazione
- **getInitParameter(name)** restituisce il valore del parametro passato per nome

Oggetto request

Rappresenta la richiesta alla pagina JSP. È il parametro request passato al metodo service() della servlet

Consente l'accesso a tutte le informazioni relative alla richiesta HTTP: indirizzo di provenienza, URL, headers, cookie, parametri, ecc.

```
<% String xStr = request.getParameter("num");
try {
    long x = Long.parseLong(xStr); %>
    Fattoriale: <%= x %>! = <%= fact(x) %>
}
catch (NumberFormatException e) { %>
    Il parametro <b>num</b>non contiene un valore intero.
<% } %>
```

Oggetto **HttpServletRequest**, creato dal **container web** per ogni richiesta JSP, associato alle richieste, questo oggetto viene utilizzato per ottenere informazioni sulla richiesta come parametro. Quando il client richiede una pagina, il motore JSP crea un nuovo oggetto per rappresentare tale richiesta.

Index.jsp :

```
<form action="Welcome.jsp">

    UserName : <input type="text" name="uname">

    Email : <input type="text" name="Email">

    <input type="submit" value="Login"><br/>

</form>
```

Welcome.jsp:

```
<%= "Welcome "+request.getParameter("uname")+" Email : "+request.getParameter("Email")%>
```

Oggetto response

Oggetto legato all'I/O della pagina JSP. Consente di inserire nella risposta diverse informazioni: content type, header di risposta, URL Rewriting, cookie.

```

<%response.setDateHeader("Expires", 0);
 response.setHeader("Pragma", "no-cache");
 if (request.getProtocol().equals("HTTP/1.1"))
 {
    response.setHeader("Cache-Control", "no-cache");
 }
%>

```

Oggetto ***HttpServletResponse***, creato dal ***container web*** per ogni richiesta JSP, associato alle risposte. Tutte le risposte implementano l'interfaccia ***ServletResponse***, che definisce i metodi per:

- Recuperare un ***flusso di output*** per utilizzare ed inviare dati al client
- Indicare il ***tipo di contenuto***(es text/html)
- Indica se bufferizzare l'output col metodo ***setBufferSize(int)***
- Imposta le informazioni di localizzazione come le ***impostazioni internazionali e codifica caratteri***

Es: <% response.sendRedirect("http://www.w3adda.com");%>

Oggetto out

Oggetto ***PrintWriter*** utilizzato per stampare l'output, legato all'I/O della pagina JSP. È uno stream di caratteri e rappresenta lo stream di output della pagina.

Es: <body>

```

<% out.print("Welcome to W3adda.com");%>

</body>

```

```

<p>Conto delle uova
<%
    int count = 0;
    while (carton.hasNext())
    {
        count++;
        out.print(".");
    }
%>
<br/>
Ci sono <%= count %> uova.
</p>

```

Metodi:

- ***isAutoFlush()*** dice se output buffer è stato impostato in modalità autoFlush o meno
- ***getBufferSize()*** restituisce dimensioni del buffer
- ***getRemaining()*** indica quanti byte liberi ci sono nel buffer
- ***clearBuffer()*** ripulisce il buffer
- ***flush()*** forza l'emissione del contenuto del buffer
- ***close()*** fa flush e chiude lo stream

Oggetto session

Oggetto che fornisce informazioni sul contesto di esecuzione della JSP. Rappresenta la sessione corrente per un utente.

L'attributo ***session*** della direttiva ***page*** deve essere true affinché JSP partecipi alla sessione.

```

<% UserLogin userData = new UserLogin(name, password);
   session.setAttribute("login", userData);
%>
<%UserLogin userData=(UserLogin)session.getAttribute("login");
 if (userData.isGroupMember("admin")) {
   session.setMaxInactiveInterval(60*60*8);
 } else {
   session.setMaxInactiveInterval(60*15);
 }
%>

```

Oggetto **HttpSession** associato alle richieste, che fornisce un modo per identificare un utente attraverso più richieste di pagina, possiamo aggiungere e rimuovere attributi ed ottenere informazioni sulla sessione utente.

Impostazione degli attributi in sessione: `session.setAttribute("user", name);`

Prelevare informazioni dalla sessione: `String name=(String)session.getAttribute("user");`

index.jsp:

```

<body>

<form action="session.jsp">

 Name : <input type="text" name="uname">

 Email : <input type="text" name="email">

 <input type="submit" value="Click"><br/>

</form>

```

</body>

Session.jsp:

```

<body>

<%
 String name = request.getParameter("uname");

 String email=request.getParameter("email");

 out.print("Welcome " + name+ " Email :" +email);

 session.setAttribute("user", name); //set attribute in session

 session.setAttribute("email", email);

%>

```

`Next Page`

</body>

Session_second.jsp:

```

<body>

<%
 String name=(String)session.getAttribute("user"); //Getting Session Attribute

```

```

String email=(String)session.getAttribute("email");
out.print("Welcome to Session Page: Name"+name+ " Email"+email);
%>
</body>

```

Oggetto application

Oggetto che fornisce informazioni su contesto di esecuzione della JSP (è **ServletContext**). Rappresenta la Web application a cui JSP appartiene.

Consente di interagire con l'ambiente di esecuzione:

- fornisce la versione di JSP Container
- garantisce l'accesso a risorse server-side
- permette accesso ai parametri di inizializzazione relativi all'applicazione
- consente di gestire gli attributi di un'applicazione

Oggetto **ServletContext** associato al contesto dell'applicazione, viene **creato una sola volta** dal web container quando l'applicazione viene eseguita. Abbiamo bisogno di inizializzare il parametro nel **web.xml** (serve per creare un mapping tra le servlet, ma le webannotation le creano da sole) e può essere utilizzato in tutte le pagine JSP.

Welcome.jsp:

```

<%
    String siteName = application.getInitParameter("SiteName");
    out.print("Welcome " + siteName);
%>

```

Oggetto pageContext

Oggetto che fornisce informazioni sul contesto di esecuzione della pagina JSP. Rappresenta l'insieme degli oggetti impliciti di una JSP, e trasferimento del controllo ad altre pagine.

Utilizzando questo oggetto è possibile lavorare con gli attributi (find, get, set, remove) in qualsiasi livello

1. JSP Page – Scope: **PAGE_CONTEXT** (default)
2. HTTP Request – Scope: **REQUEST_CONTEXT**
3. HTTP Session – Scope: **SESSION_CONTEXT**
4. Application Level – Scope: **APPLICATION_CONTEXT**

```

<% pageContext.setAttribute("role", "manager", PageContext.SESSION_SCOPE); %>
<% pageContext.getAttribute("mail", PageContext.APPLICATION_SCOPE); %>

```

Oggetto **PageContext** che fornisce l'accesso a tutti gli spazi dei nomi associati alle pagine JSP. È una classe astratta che definisce alcuni campi come:

Index.jsp:

```

<body>
<form action="PageContext.jsp">
    Name : <input type="text" name="uname">
    Email : <input type="text" name="email">

```

```

<input type="submit" value="Click">
</form>
</body>
pageContext.jsp:
<body>
<%
String name = request.getParameter("uname");
String email=request.getParameter("email");
out.print("Welcome " + name+ " Email :" +email);
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
%>
<a href = "pagecontext_second.jsp">Next Page</a>
</body>
pageContext_Second.jsp:
<body>
<%
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
%>
</body>

```

Oggetto exception

Oggetto connesso alla gestione degli errori. Rappresenta l'eccezione che non viene gestita da nessun blocco catch.

```

<%@ page isErrorPage="true" %>
<h1>Attenzione!</h1>
E' stato rilevato il seguente errore:<br/>
<b><%= exception %></b><br/>
<%
    exception.printStackTrace(out);
%>

```

Oggetto **Exception** consente l'accesso ai dati delle eccezioni da parte del JSP, è un'istanza di **java.lang.Exception**.

Quando si è verificata la condizione di errore, questo oggetto implicito viene utilizzato per generare una risposta appropriata. In JSP possiamo eseguire la gestione delle eccezioni in:

#Page directive, creando una pagina JSP: <%@ page isErrorPage="exception.jsp" %>

exception.jsp:

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page isErrorPage="true" %> -----

```

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>W3Adda Exception Page</title>
  </head>
  <body>
    <h3>Sorry an exception occurred!</h3>
    Exception is: <%= exception%>
  </body>
</html>
```

processingPage.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page errorPage="exception.jsp" %> -----
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>W3Adda JSP Page</title>
  </head>
  <body>
    <%
      String num1 = request.getParameter("num1");
      String num2 = request.getParameter("num2");
      int a = Integer.parseInt(num1);
      int b = Integer.parseInt(num2);
      int c = a / b;
      out.print("The answer is: " + c);
    %>
  </body>
</html>
```

Domanda 16 (punteggio: da 0 a 4) Scrivere una jsp che riceva un indirizzo di posta elettronica da ciascun client che vi accede e visualizzi gli indirizzi degli ultimi due accessi incluso quello appena ricevuto.

```
<html>
<head>
<title>test.jsp</title>
</head>
<body>
<%
String uEmail = request.getParameter("email");
if(uEmail != null) {
    String aEmail = (String)application.getAttribute("LAST_EMAIL");
    application.setAttribute("LAST_EMAIL", uEmail);
%>
Ultima email: <%= uEmail %><br>
Penultima email: <%= aEmail %>
<% } else { %>
<form action="test.jsp">
Inserisci email:<br>
<input type="text" name="email" value=""><br>
<input type="submit" value="Invia">
</form>
<% } %>
</body>
</html>
```

JavaBeans

Un **JavaBean**, o semplicemente **bean**, non è altro che **una classe Java** dotata di alcune caratteristiche particolari:

- Classe **public**
- Ha un **costruttore** public di **default** (*senza argomenti*)
- Espone proprietà, sotto forma di coppie di metodi di accesso (accessors) costruiti secondo una convenzione standard per i nomi dei metodi (**get... set...**):
 - La proprietà **prop** è definita da due metodi **getProp()** e **setProp()**
 - Il tipo del parametro di **setProp(...)** e del valore di ritorno di ... **getProp()** devono essere uguali e rappresentano il tipo della proprietà
 - Se definiamo solo il metodo **get** avremo una proprietà in sola lettura (read-only)
 - Le proprietà di tipo boolean seguono una regola leggermente diversa: metodo di lettura ha la forma **isProp()**
 - Ci sono anche proprietà indicizzate per rappresentare collezioni di valori
- Espone eventi con metodi di registrazione che seguono regole precise(listener, eventi)

```
import java.util.*  
public class CurrentTimeBean  
{  
    private int hours;  
    private int minutes;  
    public CurrentTimeBean()  
    {  
        Calendar now = Calendar.getInstance();  
        this.hours = now.get(Calendar.HOUR_OF_DAY);  
        this.minutes = now.get(Calendar.MINUTE);  
    }  
    public int getHours()  
    { return hours; }  
    public int getMinutes()  
    { return minutes; }  
}
```

JSP prevedono una serie di tag per agganciare un **bean** e utilizzare le sue proprietà all'interno della pagina

JavaBean sono classi che encapsulano molti oggetti in un singolo, Contiene metodi di costruzione come Getter e Setter, impostando ed ottenendo dati.

simpleBean.java:

```
public class simpleBean {  
  
    private int id;  
  
    private String name;  
  
    public simpleBean() { }  
  
    public void setId(int id) {  
  
        this.id = id;  
    }  
  
    public int getId() {  
  
        return id;  
    }  
  
    public void setName(String name) {
```

```
this.name = name;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
}
```

testBean.java:

```
public class testBean {
```

```
    public static void main(String args[]) {
```

```
        simpleBean sb = new simpleBean(); //object is created
```

```
        sb.setName("w3addaa"); //setting value to the object
```

```
        System.out.println(sb.getName()); //getting and printing the output
```

```
}
```

```
}
```

DAO (Data Access Object)

Il pattern DAO rappresenta un possibile modo di separare:

- *logica di business (es: Servlet, JSP, ...)*
- *logica di persistenza (es: r/w su DB, ...)*

Solo gli oggetti previsti dal pattern DAO hanno il permesso di “vedere” il DB ed espongono metodi di accesso per tutti gli altri componenti

Per ogni classe **MyData** che rappresenta una entità del dominio (*bean*), si definisce una classe **MyDataDAO** con i seguenti metodi:

un metodo **doSave(MyData o)** che salva i dati dell'oggetto corrente nel database:

- il metodo esegue una istruzione **SQL insert**

un metodo **doSaveOrUpdate(MyData o)** che salva i dati dell'oggetto corrente nel database:

- il metodo esegue una istruzione **SQL update** o **insert** a seconda che l'oggetto corrente esista già o meno nel database

un metodo **doDelete(X key)** che cancella dal database i dati dell'oggetto corrente:

- Si può usare anche **doDelete(MyData a)**

un metodo **doRetrieveByKey(X key)** che recupera i dati in base alla chiave:

- restituisce un oggetto istanza di **MyData** i cui dati sono letti dal database

uno o più metodi **doRetrieveByCond(...)** che restituiscono una collezione di oggetti istanza della classe **MyData** che soddisfano una qualche condizione

uno metodo **doRetrieveAll(...)** che restituisce tutta la collezione di oggetti istanza della classe **MyData**

<https://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

*Per effettuare una query parametrica si usa **PreparedStatement**, mentre per accedere ai dati recuperati dal database dopo l'esecuzione di una query si usa **ResultSet**.*

CSS (Cascading Style Sheets(Fogli di stile)) - https://www.w3schools.com/css/css_intro.asp

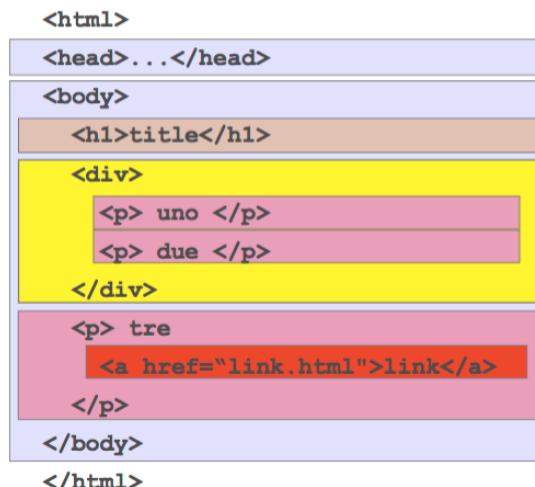
Si tratta di una serie di regole che permettono di definire l'aspetto (lo stile) che devono assumere gli elementi sulla pagina. Dimensioni, colori, animazioni, ogni caratteristica visuale può essere manipolata. Questi file possono essere riusabili, basta cambiare il CSS e può essere presentato correttamente in modo ottimale su dispositivi diversi (es. PC e palmari). Si può dividere il lavoro fra chi gestisce il contenuto e chi si occupa della parte grafica.

Ridurre i tempi di scaricamento delle pagine: una pagina che usa i CSS è meno della metà di una pagina che usa la formattazione con tag HTML, inoltre se il file CSS è condiviso da più pagine, viene scaricato una volta sola.

Ripulire il codice HTML: eliminare l'uso di estensioni non proprietarie. *Separano la struttura dalla presentazione (view) del documento HTML.*

La struttura in HTML

```
<!doctype html>
<html>
<head>
<title>Hello World</title>
</head>
<body>
<h1>Hello World!</h1>
<p>Usiamo i CSS</p>
</body>
</html>
```



Un documento HTML può essere visto come un insieme di blocchi (contenitori) sui quali si può agire con stili diversi.

Ogni tag HTML definisce un blocco.

Tre modi per inserire CSS

Esistono tre modi per inserire un foglio di stile:

- **Foglio di stile esterno**, puoi cambiare l'aspetto di un intero sito web cambiando un solo file.

Ogni pagina deve includere un riferimento al file del foglio di stile esterno all'interno dell'elemento <link>, quest'ultimo rientrerà in <head>

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

```
mystyle.css
body { background-color: lightblue; }
h1 {
    color: navy;
    margin-left: 20px;
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_howto_external

- **Foglio di stile interno**, può essere usato se una singola pagina ha uno stile unico. Gli stili interni sono definiti all'interno dell'elemento <style>, all'interno della sezione <head> di una pagina HTML.

```
<!DOCTYPE html>
<html>
<head>
<style>
body { background-color: linen; }
h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

This is a heading

This is a paragraph.

https://www.w3schools.com/css/tryit.asp?filename=trycss_howto_internal

- **Stile in linea** può essere usato per applicare uno stile unico per un singolo elemento. Per usare gli stili in linea, aggiungi l'attributo style all'elemento pertinente.

```
<!DOCTYPE html>
<html>
<body>
<h1 style="color:blue; margin-left:30px;">
This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_howto_inline

This is a heading

This is a paragraph.

Se alcune proprietà sono state definite per lo stesso selettori (elemento) in fogli di stile diversi, verrà utilizzato il valore **dell'ultimo foglio di stile letto**.

Ordine a cascata

Tutti gli stili di una pagina "si sovrappongono" in un nuovo foglio di stile "virtuale" secondo le regole:

1. **Stile in linea** (all'interno di un elemento HTML)
2. **Fogli di stile esterni e interni** (nella sezione della testa)
3. Browser predefinito

Proprietà (Stenografia)

Nelle dichiarazioni è possibile far uso di **proprietà singole** o in forma abbreviata (**shorthand properties**)

- Le **proprietà singole** permettono di definire un singolo aspetto di stile
- Le **shorthand properties** consentono invece di definire un insieme di aspetti, correlati fra di loro usando una sola proprietà

Per esempio, ogni elemento permette di definire un margine rispetto a quelli adiacenti usando quattro proprietà: **margin-top, margin-right, margin-bottom, margin-left**

Utilizziamo le proprietà singole applicandole ad un paragrafo:

```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
```

Lo stesso risultato si può ottenere usando la proprietà in forma abbreviata:

```
p {
  margin: 25px 50px 75px 100px;
}
```

Valori

Numeri interi e reali: “.” come separatore decimale

Grandezze: usate per lunghezze orizzontali e verticali, un numero seguito da una unità di misura

Unità di misura relative:

- **em**: è relativa alla dimensione del font in uso (es. se il font ha corpo 12pt, 1em varrà 12pt, 2em varranno 24pt)
- **px**: pixel, sono relativi al dispositivo di output e alle impostazioni del computer dell'utente Unità di misura assolute
- **in**: pollici; (1 in = 2.54 cm)

- **cm**: centimetri
- **mm**: millimetri
- **pt**: punti tipografici (1/72 di pollice)
- **pc**: pica = 12 punti

Percentuali: percentuale del valore che assume la proprietà stessa nell'elemento padre; un numero seguito da %

URL assoluti o relativi; si usa la notazione **url(percоро)**

Stringhe: testo delimitato da apici singoli o doppi

Ereditarietà

È un meccanismo di tipo differenziale simile per certi aspetti all'ereditarietà nei linguaggi ad oggetti, si basa sui blocchi annidati di un documento HTML. **Uno stile applicato ad un blocco esterno si applica anche ai blocchi in esso contenuti**

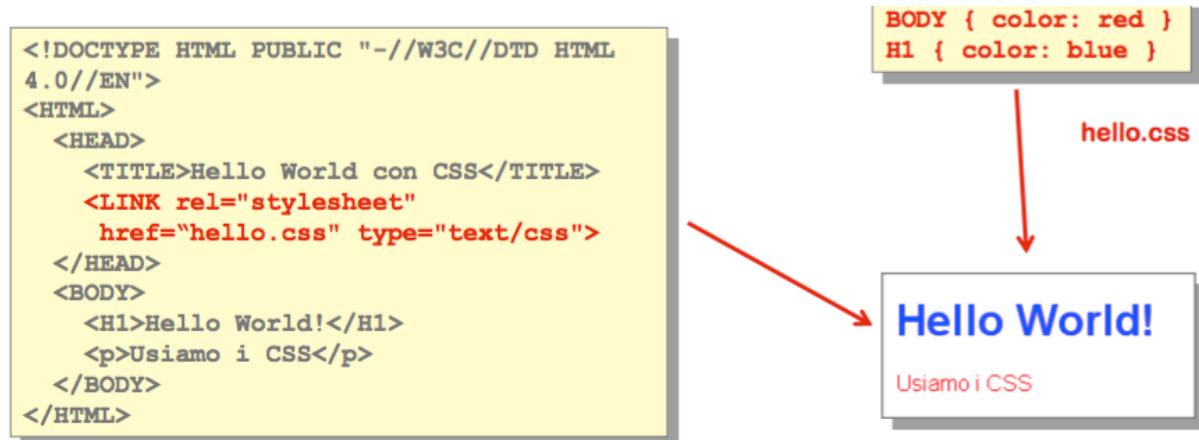
In un blocco interno:

- Si possono definire stili aggiuntivi
- Si possono ridefinire stili definiti in un blocco più esterno (è una sorta di overriding)

Lo stesso ragionamento si può esprimere in termini di **DOM**

Un nodo figlio eredita gli stili dei nodi che si trovano sul ramo da cui discende

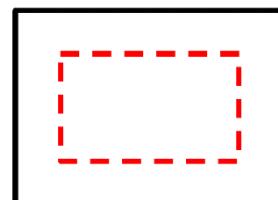
L'elemento **<p>Usiamo i CSS</p>** non ridefinisce il colore del testo e quindi eredita da **<body>** (viene mostrato in rosso), l'elemento **<h1>Hello World</h1>** ridefinisce lo stile e quindi appare in blu



Limitazioni dell'ereditarietà

Non tutte le proprietà sono soggette al meccanismo dell'ereditarietà, in generale non vengono ereditate quelle che riguardano la formattazione del box model

- Il box è il riquadro che circonda ogni elemento:



Sintassi - https://www.w3schools.com/css/css_syntax.asp

Una serie di **regole CSS** è composta da un **selettore** e un **blocco di dichiarazione**:



- Il **selettore** punta all'elemento HTML che desideri applicare allo stile.
 - Ogni **dichiarazione** include un nome di proprietà CSS e un valore, separati da due punti.
 - Una **dichiarazione CSS** termina sempre con un punto e virgola e i blocchi di dichiarazione sono circondati da parentesi graffe. https://www.w3schools.com/css/tryit.asp?filename=trycss_syntax1

Selettori

Il selettori serve per collegare uno stile agli elementi a cui deve essere applicato, e possono essere:

- **Selettore universale**, identifica qualunque elemento: * `{color: red}`

- **Selettori di tipo:** si applicano a tutti gli elementi di tipo (ad es. tutti i `<p>`)

```
p {  
    text-align: center;  
    color: red;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_syntax\_element
```

- **Classi:** applicato a tutti gli elementi che presentano l'attributo `class="center"`

```
.center {  
  text-align: center;  
  color: red;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_syntax\_class
```

Puoi anche specificare che solo gli elementi HTML specifici dovrebbero essere influenzati da una classe.

```
p.center {  
    text-align: center;  
    color: red;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_syntax\_element\_class
```

- **Identifieri:** si applica all'elemento che presenta l'attributo `id="para1"`

```
#para1 {  
    text-align: center;  
    color: red;  
}
```

- **Selettori di raggruppamento**: se hai elementi con le stesse definizioni di stile

```
h1, h2, p {  
    text-align: center;  
    color: red;  
}
```

I **selettori di tipo** si possono **combinare** con quelli di classe e di identificatore:

tipo_elemento nome classe { ... } oppure *tipo_elemento#nome id { ... }*

```
p#footer {  
    color: red;  
}  
  
This selects a <p> element if it has the id "footer".
```

Un **commento** CSS inizia con `/*` e termina con `*/`. I commenti possono anche estendersi su più righe.

Combinatori - https://www.w3schools.com/css/css_combinators.asp

Un selettori CSS può contenere più di un semplice selettori. Tra i selettori semplici, possiamo includere un combinatore, ci sono quattro diversi:

- **selettore discendente (spazio)**, corrisponde a tutti gli elementi che discendono da un elemento specificato. Seleziona tutti gli elementi `<p>` all'interno degli elementi `<div>`:

```
div p {  
    background-color: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_element
```

- **selettore di bambini (>)**, seleziona tutti gli elementi che sono figli immediati di un elemento specificato.

Seleziona tutti gli elementi `<p>` che sono figli immediati di un elemento `<div>`:

```
div > p {  
    background-color: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_gt
```

- **selettore fratello adiacente (+)**, seleziona tutti gli elementi che sono i fratelli adiacenti di un elemento specificato. Gli elementi di pari livello devono avere lo stesso elemento principale e "adiacente" significa "**immediatamente successivo**". Seleziona tutti gli elementi `<p>` posizionati dopo gli elementi `<div>`:

```
div + p {  
    background-color: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_pluss
```

- **selettore di fratello generale (~)**, seleziona tutti gli elementi che sono fratelli di un elemento specificato.

Seleziona tutti gli elementi `<p>` che sono fratelli di elementi `<div>`:

```
div ~ p {  
    background-color: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_tilde
```

Pseudo-classi - https://www.w3schools.com/css/css_pseudo_classes.asp

Una **pseudo-classe** è usata per definire uno stato speciale di un elemento, può essere utilizzato per:

- Disegna un elemento quando un utente lo sostituisce
- Stile visitato e collegamenti non visitati in modo diverso
- Disegna un elemento quando si concentra

La sintassi delle pseudo-classi:

```
selector:pseudo-class {  
    property:value;  
}
```

I collegamenti possono essere visualizzati in diversi modi, con l'utilizzo di **anchor <a>**:

```
/* unvisited link */  
a:link {  
    color: #FF0000;  
}  
/* visited link */  
a:visited {  
    color: #00FF00;  
}  
/* mouse over link */  
a:hover {  
    color: #FF00FF;  
}  
/* selected link */  
a:active {  
    color: #0000FF;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss_link
```

Nota: `a:hover` DEVE venire dopo `a:link` e `a:visited` nella definizione CSS per essere efficace! `a:active` DEVE venire dopo `a:hover` nella definizione CSS per essere efficace...

Le pseudo-classi possono essere combinate con le classi CSS.

Quando si passa il mouse sul collegamento nell'esempio, cambierà colore:

```
a.highlight:hover {  
  color: #ff0000;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_pseudo-class

Un esempio di utilizzo della **:hover** pseudo-classe su un elemento **<div>**:

```
div:hover {  
  background-color: blue;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_pseudo-class_hover_div

Passaggio rapido del suggerimento semplice

Passa il mouse su un elemento **<div>** per mostrare un elemento **<p>** (come un suggerimento):

```
p {  
  display: none;  
  background-color: yellow;  
  padding: 20px;  
}  
  
div:hover p {  
  display: block;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_pseudo-class_hover_tooltip

La pseudo-classe: first-child

La **:first-child** pseudo-classe corrisponde a un elemento specificato che è il primo figlio di un elemento.

Solo la prima occorrenza dell'elemento avrà lo stile descritto.

```
p:first-child {  
  color: blue;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_first-child1

Il selettore corrisponde al primo elemento **<i>** in tutti gli elementi **<p>**:

```
p i:first-child {  
  color: blue;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_first-child2

Il selettore corrisponde a tutti gli elementi **<i>** negli elementi **<p>** che sono il primo figlio di un altro elemento:

```
p:first-child i {  
  color: blue;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_first-child3

Pseudo-elementi - https://www.w3schools.com/css/css_pseudo_elements.asp

Uno pseudo-elemento CSS è usato per dare uno stile alle parti specificate di un elemento, può essere utilizzato per:

- Stile la prima lettera, o linea, di un elemento
- Inserisci il contenuto prima o dopo il contenuto di un elemento

La sintassi degli pseudo-elementi:

```
selector::pseudo-element {  
  property:value;  
}
```

Il doppio punto sostituisce la notazione a due punti per gli pseudo-elementi in CSS3.

Lo pseudo-elemento ::first-line

Lo pseudo-elemento **::first-line** è usato per aggiungere uno stile speciale alla prima riga di un testo (può essere applicato solo agli elementi a livello di blocco **<div>**).

```
p::first-line {  
    color: #ff0000;  
    font-variant: small-caps;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_firstline
```

Lo pseudo-elemento ::first letter

Lo pseudo-elemento **::first-letter** è usato per aggiungere uno stile speciale alla prima lettera di un testo (può essere applicato solo agli elementi a livello di blocco **<div>**).

```
p::first-letter {  
    color: #ff0000;  
    font-size: xx-large;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_firstletter
```

Pseudo-elementi e classi

Gli pseudo-elementi possono essere combinati con le classi CSS.

```
p.intro::first-letter {  
    color: #ff0000;  
    font-size: 200%;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_pseudo-element
```

Pseudo-elementi multipli

Diversi pseudo-elementi possono anche essere combinati.

```
p::first-letter {  
    color: #ff0000;  
    font-size: xx-large;  
}  
p::first-line {  
    color: #0000ff;  
    font-variant: small-caps;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_firstline\_letter
```

Pseudo-elementi ::before

Lo pseudo-elemento **::before** può essere usato per inserire del contenuto prima di un elemento.

```
h1::before {  
    content: url(smiley.gif);  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_before
```

Pseudo-elementi ::after

Lo pseudo-elemento **::after** può essere usato per inserire del contenuto dopo un elemento.

```
h1::after {  
    content: url(smiley.gif);  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_after
```

Pseudo-elementi ::selection

Lo pseudo-elemento **::selection** corrisponde alla porzione di un elemento che è selezionato da un utente.

Le seguenti proprietà CSS possono essere applicate a **::selection: color, background, cursor, e outline**.

```
::selection {  
    color: red;  
    background: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss3\_selection
```

Display - https://www.w3schools.com/css/css_display_visibility.asp

Ogni elemento HTML ha un valore di visualizzazione predefinito a seconda del tipo di elemento. Il valore di visualizzazione predefinito per la maggior parte degli elementi è **block o inline**.

Block-level Elements

Un elemento a livello di blocco inizia sempre su una nuova riga e occupa l'intera larghezza disponibile.

- <Div>
- <h1> - <h6>
- <P>
- <Form>
- <Section>

Inline Elements

Un elemento in linea non inizia su una nuova riga e occupa solo la larghezza necessaria.

-
- <a>
-

display:none

Comunemente usato con JavaScript per nascondere e mostrare elementi senza eliminarli e ricrearli.

```
h1.hidden {  
    display: none;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_display\_none
```

visibility: hidden

Nasconde un elemento, quest'ultimo occuperà ancora lo stesso spazio di prima. *L'elemento sarà nascosto, ma influenzerà il layout.*

```
h1.hidden {  
    visibility: hidden;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_visibility\_hidden
```

Display: inline-block - https://www.w3schools.com/css/css_inline-block.asp

Rispetto a **display: inline**, la principale differenza è che **display: inline-block** consente di impostare una larghezza e un'altezza sull'elemento.

I margini superiore e inferiore / paddings sono rispettati, ma con **display: inline** essi non lo sono.

Rispetto a **display: block**, la differenza principale è che **display: inline-block** non aggiunge un'interruzione di riga dopo l'elemento, quindi l'elemento può sedersi accanto ad altri elementi.

```
span.b {  
    display: inline-block;  
    width: 100px;  
    height: 100px;  
    padding: 5px;  
    border: 1px solid blue;
```

```
background-color: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_inline-block\_span1
```

Bordi - https://www.w3schools.com/css/css_border.asp

Le **border** proprietà CSS consentono di specificare lo stile, la larghezza e il colore del bordo di un elemento.

La **border-style** proprietà specifica quale tipo di bordo visualizzare e può avere da uno a quattro valori (per il bordo superiore, il bordo destro, il bordo inferiore e il bordo sinistro). Sono ammessi i seguenti valori:

- **dotted** - Definisce un bordo tratteggiato, `p.dotted {border-style: dotted;}`
- **dashed** - Definisce un bordo tratteggiato, `p.dashed {border-style: dashed;}`
- **solid** - Definisce un bordo solido, `p.solid {border-style: solid;}`
- **double** - Definisce un doppio bordo, `p.double {border-style: double;}`
- **groove** - Definisce un bordo scanalato 3D. L'effetto dipende dal valore del colore del bordo
- **ridge** - Definisce un bordo increspato in 3D. L'effetto dipende dal valore del colore del bordo
- **inset** - Definisce un bordo interno 3D. L'effetto dipende dal valore del colore del bordo
- **outset** - Definisce un bordo esterno 3D. L'effetto dipende dal valore del colore del bordo
- **none** - Non definisce nessun confine
- **hidden** - Definisce un bordo nascosto

https://www.w3schools.com/css/tryit.asp?filename=trycss_border-style

Larghezza del bordo

La **border-width** proprietà specifica la larghezza dei quattro bordi.

La larghezza può essere impostata come dimensione specifica. O utilizzando uno dei tre valori predefiniti: sottile, medio o spesso (**thin**, **medium**, or **thick**). Può avere da uno a quattro valori.

```
p.one {  
    border-style: solid;  
    border-width: 5px;  
}  
p.two {  
    border-style: solid;  
    border-width: medium;  
}  
p.three {  
    border-style: solid;  
    border-width: 2px 10px 4px 20px;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_border-width

Colore del bordo

La **border-color** proprietà viene utilizzata per impostare il colore dei quattro bordi. Può avere da uno a quattro valori (per il bordo superiore, il bordo destro, il bordo inferiore e il bordo sinistro).

Se **border-color** non è impostato, eredita il colore dell'elemento.

```
p.one {  
    border-style: solid;  
    border-color: red;  
}
```

```

p.two {
  border-style: solid;
  border-color: green;
}
p.three {
  border-style: solid;
  border-color: red green blue yellow;
}

```

https://www.w3schools.com/css/tryit.asp?filename=trycss_border-color1

Border - Singoli lati

In CSS, ci sono anche proprietà per specificare ciascuno dei bordi (in alto, a destra, in basso e a sinistra):

```

p {
  border-top-style: dotted;
  border-right-style: solid;
  border-bottom-style: dotted;
  border-left-style: solid;
}

```

https://www.w3schools.com/css/tryit.asp?filename=trycss_border-side

Margini - https://www.w3schools.com/css/css_margin.asp

Le **margin** vengono utilizzate per creare lo spazio attorno agli elementi, al di fuori dei confini definiti.

https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_sides

Margine - Lati individuali

I CSS hanno proprietà per specificare il margine per ciascun lato di un elemento:

```

p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}

```

https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_sides

Tutte le proprietà del margine possono avere i seguenti valori:

- **auto**: il browser calcola il margine
- **length** - specifica un margine in px, pt, cm, ecc.
- **%** : specifica un margine in% della larghezza dell'elemento contenitore
- **inherit**: specifica che il margine deve essere ereditato dall'elemento padre

Suggerimento: i valori negativi sono consentiti.

Il valore automatico

È possibile impostare la proprietà del margine **auto** per centrare orizzontalmente l'elemento all'interno del relativo contenitore. L'elemento prenderà quindi la larghezza specificata e lo spazio rimanente sarà diviso equamente tra i margini sinistro e destro:

```

div {
  width: 300px;
  margin: auto;
  border: 1px solid red;
}

```

https://www.w3schools.com/css/tryit.asp?filename=trycss_margin_auto

Il valore ereditario

Consente di ereditare il margine sinistro dell'elemento `<p class = "ex1">` dall'elemento padre (`<div>`):

```
div {  
    border: 1px solid red;  
    margin-left: 100px;  
}  
p.ex1 {  
    margin-left: inherit;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_margin-left\_inherit
```

Padding - https://www.w3schools.com/css/css_padding.asp

Le **padding** vengono utilizzate per generare spazio attorno al contenuto di un elemento, all'interno di qualsiasi bordo definito.

Imbottitura - Lati singoli

I CSS hanno proprietà per specificare il riempimento per ciascun lato di un elemento:

```
div {  
    padding-top: 50px;  
    padding-right: 30px;  
    padding-bottom: 50px;  
    padding-left: 80px;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_padding\_sides
```

Tutte le proprietà di **padding** possono avere i seguenti valori:

- **length** - specifica un padding in px, pt, cm, ecc.
- **%** : specifica un riempimento in% della larghezza dell'elemento contenitore
- **inherit** - specifica che il padding dovrebbe essere ereditato dall'elemento genitore

Imbottitura e larghezza dell'elemento

La **width** specifica la larghezza dell'area del contenuto dell'elemento. L'area del contenuto è la parte all'interno del padding, del bordo e del margine di un elemento. Quindi, se un elemento ha una larghezza specificata, il riempimento aggiunto a quell'elemento verrà aggiunto alla larghezza totale dell'elemento. Questo è spesso un risultato indesiderato.

```
div {  
    width: 300px;  
    padding: 25px;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_padding\_width
```

Per mantenere la larghezza a 300 px, indipendentemente dalla quantità di padding, è possibile utilizzare la **box-sizing**. Ciò fa sì che l'elemento mantenga la sua larghezza; se si aumenta il padding, lo spazio contenuto disponibile diminuirà.

```
div {  
    width: 300px;  
    padding: 25px;  
    box-sizing: border-box;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_padding\_width2
```

Altezza e Larghezza - https://www.w3schools.com/css/css_dimension.asp

Le proprietà **height** e **width** vengono utilizzate per impostare l'altezza e la larghezza di un elemento, può essere impostato su automatico oppure essere specificato in valori di lunghezza , come px, cm, ecc. O in percentuale (%) del blocco contenitore.

```
div {  
    height: 200px;  
    width: 50%;  
    background-color: powderblue;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_dim_height_width2

Impostazione della larghezza massima

La **max-width** viene utilizzata per impostare la larghezza massima di un elemento.

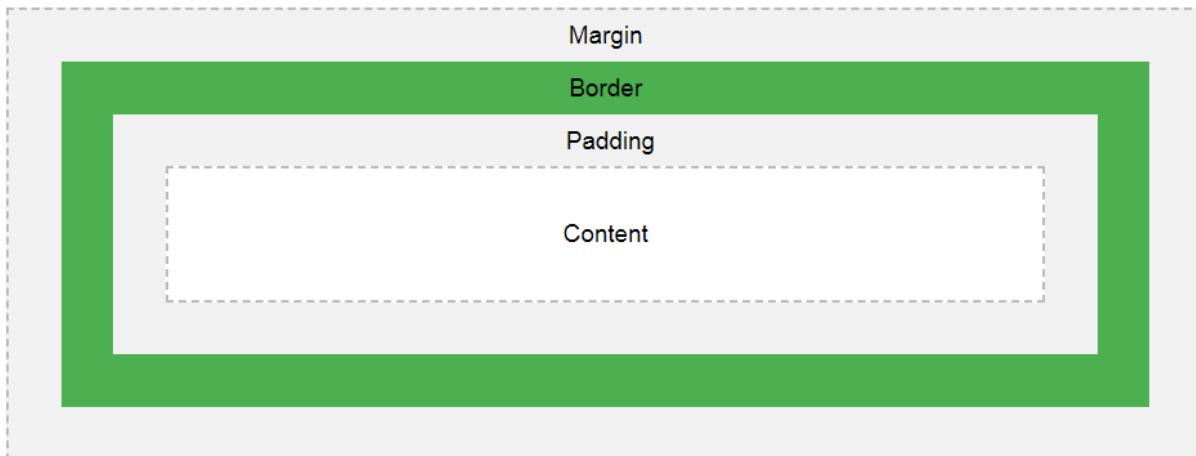
Nota: il valore della proprietà **max-width** sostituisce **width**.

```
div {  
    max-width: 500px;  
    height: 100px;  
    background-color: powderblue;  
}
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_dim_max_width

Box Model - https://www.w3schools.com/css/css_boxmodel.asp

Tutti gli elementi HTML possono essere considerati come scatole. Nei CSS, il termine "**box model**" viene usato quando si parla di design e layout. Il modello di box CSS è essenzialmente una scatola che avvolge ogni elemento HTML. Consiste di: **margini, bordi, spaziatura e contenuto** effettivo.

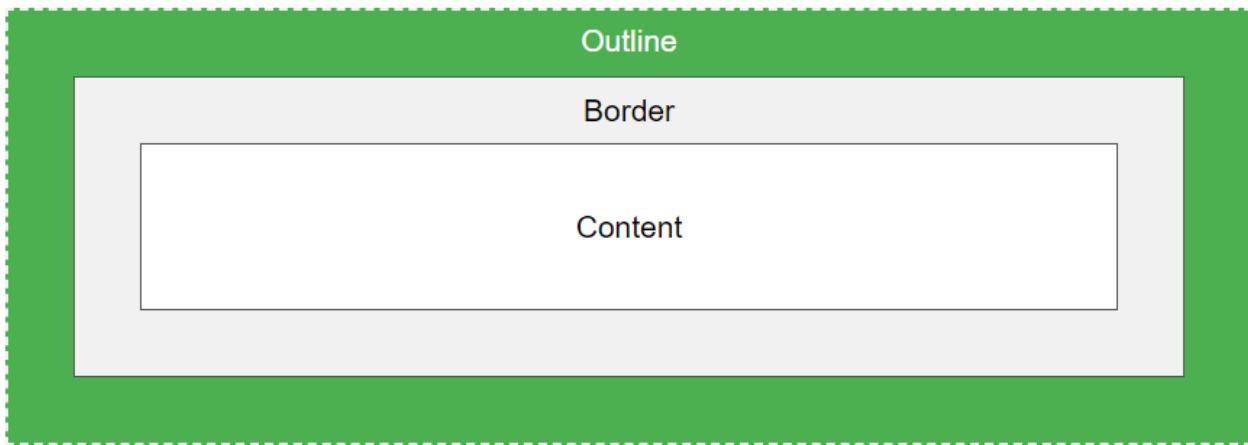


- **Contenuto** : il contenuto della casella, in cui vengono visualizzati testo e immagini
- **Riempimento** - Cancella un'area attorno al contenuto. L'imbottitura è trasparente
- **Bordo** : un bordo che circonda il riempimento e il contenuto
- **Margine** - Cancella un'area al di fuori del confine. Il margine è trasparente

https://www.w3schools.com/css/tryit.asp?filename=trycss_boxmodel

Outline (Contorno) - https://www.w3schools.com/css/css_outline.asp

L **outline** è una linea che viene disegnata attorno agli elementi, FUORI dai bordi, per rendere l'elemento "risaltante".



Il CSS ha le seguenti proprietà del profilo:

- ***outline-style***
- ***outline-color***
- ***outline-width***
- ***outline-offset***
- ***outline***

La ***outline-style*** specifica lo stile del contorno e può avere uno dei seguenti valori:

- ***dotted*** - Definisce un contorno tratteggiato, `p.dotted {outline-style: dotted;}`
- ***dashed*** - Definisce un contorno tratteggiato, `p.dashed {outline-style: dashed;}`
- ***solid*** - Definisce un contorno solido, `p.solid {outline-style: solid;}`
- ***double*** - Definisce un doppio contorno
- ***groove*** - Definisce un contorno scanalato 3D
- ***ridge*** - Definisce un contorno tridimensionale
- ***inset*** - Definisce un contorno inserto 3D
- ***outset*** - Definisce un contorno 3D
- ***none*** - Non definisce il contorno
- ***hidden*** - Definisce un contorno nascosto

https://www.w3schools.com/css/tryit.asp?filename=trycss_outline-style

La ***outline-color*** viene utilizzata per impostare il colore del contorno. Il colore può essere impostato da:

- ***nome*** - specifica un nome di colore, come "rosso"
- ***RGB***: specifica un valore RGB, come "rgb (255,0,0)"
- ***Esiadecimale***: specificare un valore esadecimale, ad esempio "# ff0000"
- ***invert*** - esegue un'inversione di colore (che garantisce che il contorno sia visibile, indipendentemente dallo sfondo colorato)

```

p.ex1 {
  border: 1px solid black;
  outline-style: solid;
  outline-color: red;
}
p.ex2 {
  border: 1px solid black;
  outline-style: double;
  outline-color: green;
}
p.ex3 {
  border: 1px solid black;
  outline-style: outset;
}

```

```
outline-color: yellow;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_outline-color
```

La **outline-width** specifica la larghezza della struttura e può avere uno dei seguenti valori:

- **sottile** (tipicamente 1px)
- **medio** (in genere 3px)
- **spesso** (tipicamente 5px)
- Una **dimensione specifica** (in px, pt, cm, em, ecc.)

```
p.ex1 {  
    border: 1px solid black;  
    outline-style: solid;  
    outline-color: red;  
    outline-width: thin;  
}  
p.ex2 {  
    border: 1px solid black;  
    outline-style: solid;  
    outline-color: red;  
    outline-width: medium;  
}  
p.ex3 {  
    border: 1px solid black;  
    outline-style: solid;  
    outline-color: red;  
    outline-width: thick;  
}  
p.ex4 {  
    border: 1px solid black;  
    outline-style: solid;  
    outline-color: red;  
    outline-width: 4px;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_outline-width
```

Outline offset

La **outline-offset** aggiunge spazio tra una struttura e il bordo / bordo di un elemento. Lo spazio tra un elemento e il suo contorno è trasparente.

```
p {  
    margin: 30px;  
    border: 1px solid black;  
    outline: 1px solid red;  
    outline-offset: 15px;  
}  
https://www.w3schools.com/css/tryit.asp?filename=trycss\_outline-offset
```

Testo - https://www.w3schools.com/css/css_text.asp

La **color** viene utilizzata per impostare il colore del testo. Il colore è specificato da:

- un nome di colore - come "red"
- un valore HEX - come "# ff0000"
- un valore RGB - come "rgb (255,0,0)"

https://www.w3schools.com/css/tryit.asp?filename=trycss_color

Allineamento del testo

La **text-align** viene utilizzata per impostare l'allineamento orizzontale di un testo.

Un testo può essere allineato a sinistra o a destra, centrato (center) o giustificato (***justified***).

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-align

Quando la text-align è impostata su "***justified***", ogni riga viene allungata in modo che ogni riga abbia larghezza uguale e i margini sinistro e destro siano diritti.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-align_all

Decorazione del testo

La ***text-decoration*** viene utilizzata per impostare o rimuovere decorazioni dal testo.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-decoration

Il valore ***text-decoration: none***, viene spesso utilizzato per rimuovere i sottotitoli dai collegamenti.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-decoration_link

Trasformazione del testo

La ***text-transform*** viene utilizzata per specificare lettere maiuscole e minuscole in un testo.

Può essere utilizzato per trasformare tutto in lettere maiuscole o minuscole o in maiuscolo la prima lettera di ogni parola.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-transform

La ***text-indent*** viene utilizzata per specificare il rientro della prima riga di un testo:

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-indent

La ***letter-spacing*** viene utilizzata per specificare lo spazio tra i caratteri in un testo.

https://www.w3schools.com/css/tryit.asp?filename=trycss_letter-spacing

La ***line-height*** viene utilizzata per specificare lo spazio tra le righe.

https://www.w3schools.com/css/tryit.asp?filename=trycss_line-height

La ***direction*** viene utilizzata per modificare la direzione del testo di un elemento.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text_direction

La ***word-spacing*** viene utilizzata per specificare lo spazio tra le parole in un testo.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text_word-spacing

La ***text-shadow*** aggiunge ombra al testo.

https://www.w3schools.com/css/tryit.asp?filename=trycss_text-shadow

FONTS - https://www.w3schools.com/css/css_fonts.asp

In CSS, ci sono due tipi di nomi di famiglia di font:

- ***famiglia generica (generic family)*** - un gruppo di famiglie di font con aspetto simile (come "Serif" o "Monospace")
- ***famiglia di caratteri (font family)*** - una specifica famiglia di font (come "Times New Roman" o "Arial")
La ***font-family*** dovrebbe contenere diversi nomi di font come sistema "fallback". Se il browser non supporta il primo carattere, prova il font successivo e così via.

La ***font-style*** è principalmente utilizzata per specificare il testo in corsivo. Questa proprietà ha tre valori:

- ***normale*** - Il testo è mostrato normalmente
- ***corsivo*** - Il testo è mostrato in corsivo

- ***oblique*** - Il testo è "pendente" (obliquo è molto simile al corsivo, ma meno supportato)

https://www.w3schools.com/css/tryit.asp?filename=trycss_font-style

La ***font-size*** imposta la dimensione del testo.

Dimensione assoluta:

- Imposta il testo su una dimensione specificata
- Non consente a un utente di modificare la dimensione del testo in tutti i browser (errato per motivi di accessibilità)
- La dimensione assoluta è utile quando è nota la dimensione fisica dell'output

Dimensione relativa:

- Imposta la dimensione relativa agli elementi circostanti
- Consente a un utente di cambiare la dimensione del testo nei browser

https://www.w3schools.com/css/tryit.asp?filename=trycss_font-size_px

La ***font-weight*** specifica il peso di un font.

https://www.w3schools.com/css/tryit.asp?filename=trycss_font-weight

Dimensione del carattere reattiva

La dimensione del testo può essere impostata con ***vw*** un'unità, che significa "larghezza della vista".

In questo modo le dimensioni del testo seguiranno le dimensioni della finestra del browser:

https://www.w3schools.com/css/tryit.asp?filename=trycss_font_responsive

La ***font-variant*** specifica se un testo deve essere o meno visualizzato in un carattere maiuscololetto.

https://www.w3schools.com/css/tryit.asp?filename=trycss_font-variant

Font Awesome Icons - https://www.w3schools.com/css/css_icons.asp

Il modo più semplice per aggiungere un'icona alla tua pagina HTML, è con una libreria di icone.

Aggiungi il nome della classe icona specificata a qualsiasi elemento HTML in linea (come **<i>** o ****).

Link - https://www.w3schools.com/css/css_link.asp

I collegamenti possono essere in stile con qualsiasi proprietà CSS (per esempio color, font-family, background, etc.).

Inoltre, i collegamenti possono essere designati in modo diverso a seconda dello ***stato*** in cui si trovano.

I quattro ***stati*** dei collegamenti sono:

- ***a:link*** - un link normale, non visitato
- ***a:visited*** - un link che l'utente ha visitato
- ***a:hover*** - un collegamento quando l'utente va sopra
- ***a:active*** - un collegamento nel momento in cui viene fatto clic

https://www.w3schools.com/css/tryit.asp?filename=trycss_link

---a: hover DEVE venire dopo a: link e a: visited / a: active DEVE venire dopo un: hover---

La ***text-decoration*** viene principalmente utilizzata per rimuovere i sottotitoli dai collegamenti.

https://www.w3schools.com/css/tryit.asp?filename=trycss_link_decoration

La **background-color** può essere utilizzata per specificare un colore di sfondo per i collegamenti.

https://www.w3schools.com/css/tryit.asp?filename=trycss_link_background

Avanzate - Pulsanti di collegamento

https://www.w3schools.com/css/tryit.asp?filename=trycss_link_advanced

Elenchi - https://www.w3schools.com/css/css_list.asp

In HTML, ci sono due tipi principali di liste:

- **elenchi non ordinati ()** - gli elementi dell'elenco sono contrassegnati da punti elenco
- **elenchi ordinati ()** - gli elementi dell'elenco sono contrassegnati con numeri o lettere

La **list-style-type** specifica il tipo di marcatore dell'elemento della lista.

https://www.w3schools.com/css/tryit.asp?filename=trycss_list-style-type_ex

: **none**; può anche essere utilizzata per rimuovere i marcatori / proiettili. Si noti che l'elenco ha anche margine e riempimento predefiniti. Per rimuovere questo, aggiungi margin:0e padding:0a o :

https://www.w3schools.com/css/tryit.asp?filename=trycss_list-style_none

La **list-style-image** specifica un'immagine come indicatore dell'elemento della lista.

https://www.w3schools.com/css/tryit.asp?filename=trycss_list-style-image

La **list-style-position** specifica la posizione degli indicatori di voci di elenco (punti elenco).

: **outside**; significa che i punti elenco saranno al di fuori della voce dell'elenco. L'inizio di ogni riga di un elemento di elenco verrà allineato verticalmente.

: **inside**; significa che i punti elenco saranno all'interno dell'elemento dell'elenco. Poiché fa parte dell'elemento della lista, sarà parte del testo e sposterà il testo all'inizio.

https://www.w3schools.com/css/tryit.asp?filename=trycss_list-style-position

Lista stilistica con colori

Possiamo anche creare liste di stili con colori, per renderli un po' più interessanti.

Qualsiasi cosa aggiunta al tag o , interessa l'intero elenco, mentre le proprietà aggiunte al tag influiscono sui singoli elementi dell'elenco:

https://www.w3schools.com/css/tryit.asp?filename=trycss_list-style_colors

Tabelle - https://www.w3schools.com/css/css_table.asp

Per specificare i bordi della tabella in CSS, utilizzare la **border**.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_border

La **border-collapse** imposta se i bordi della tabella devono essere compressi in un singolo bordo.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_border-collapse

Se vuoi solo un bordo attorno alla tabella, specifica solo la border per <table>.

La larghezza e l'altezza di una tabella sono definite da **width** e **height**.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_width

La **text-align** imposta l'allineamento orizzontale (come sinistra, destra o centro) del contenuto in <th> o <td>.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_align

La **vertical-align** imposta l'allineamento verticale (come in alto, in basso o al centro) del contenuto in <th> o <td>.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_vertical-align

Per controllare lo spazio tra il bordo e il contenuto di una tabella, usa la **padding** sugli elementi <td> e <th>

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_padding

Aggiungi la **border-bottom** a <th> e <td> per i divisori orizzontali.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_border_divider

Usa il **:hover** selettore su <tr> per evidenziare le righe della tabella con il mouse sopra.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_hover

Per le tabelle zebrate, usa il **nth-child()** selettore e aggiungi **background-color** a tutte le righe della tabella pari (o dispari).

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_striped

Specifica il colore di sfondo e il colore del testo di <th> elementi.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_color

Una tabella reattiva mostrerà una barra di scorrimento orizzontale se lo schermo è troppo piccolo per visualizzare l'intero contenuto. Aggiungi un elemento contenitore (come <div>) overflow-x:auto attorno all'elemento <table> per renderlo reattivo.

https://www.w3schools.com/css/tryit.asp?filename=trycss_table_responsive

Posizionamento - https://www.w3schools.com/css/css_positioning.asp

La **position** specifica il tipo di metodo di posizionamento utilizzato per un elemento (statico, relativo, fisso, assoluto o appiccicoso). Esistono cinque diversi valori di posizione:

- **Static**(predefinito), gli elementi posizionati statici non sono influenzati dalle proprietà superiore, inferiore, sinistra e destra.
Un elemento con **position: static**; non è posizionato in alcun modo speciale; è sempre posizionato in base al normale flusso della pagina.
https://www.w3schools.com/css/tryit.asp?filename=trycss_position_static
- **Relative**, un elemento con **position: relative**; è posizionato rispetto alla sua posizione normale.
L'impostazione delle proprietà superiore, destra, inferiore e sinistra di un elemento relativamente posizionato causerà la sua regolazione lontano dalla sua posizione normale. Gli altri contenuti non saranno adattati per adattarsi a qualsiasi spazio lasciato dall'elemento.
https://www.w3schools.com/css/tryit.asp?filename=trycss_position_relative
- **Fixed**, un elemento con **position: fixed**; è posizionato rispetto al **viewport**, il che significa che rimane sempre nello stesso posto anche se la pagina scorre. Le proprietà superiore, destra, inferiore e sinistra vengono utilizzate per posizionare l'elemento.
Un elemento fisso non lascia una lacuna nella pagina in cui normalmente si trovava.
https://www.w3schools.com/css/tryit.asp?filename=trycss_position_fixed
- **Absolute**, un elemento con **position: absolute**; è posizionato rispetto all'antenato posizionato più vicino (anziché posizionato rispetto al **viewport**, come fisso).
Però, se un elemento posizionato in modo assoluto non ha antenati posizionati, usa il corpo del documento e si sposta insieme allo scorrimento della pagina.
https://www.w3schools.com/css/tryit.asp?filename=trycss_position_absolute
- **Sticky**, Un elemento con **position: sticky**; è posizionato in base alla posizione di scorrimento dell'utente.

Un elemento adesivo passa da ***relative*** e ***fixed***, a seconda della posizione di scorrimento. Viene posizionato relativo fino a quando una determinata posizione di offset viene soddisfatta nel ***viewport***, quindi "si attacca" in posizione (come posizione: fissa).

https://www.w3schools.com/css/tryit.asp?filename=trycss_position_sticky

Gli elementi vengono quindi posizionati usando le proprietà superiore, inferiore, sinistra e destra. Tuttavia, queste proprietà non funzioneranno se ***position*** prima non viene impostata la proprietà. Funzionano anche in modo diverso a seconda del valore della posizione.

Elementi sovrapposti

Quando gli elementi sono posizionati, possono sovrapporsi ad altri elementi.

La ***z-index*** specifica l'ordine di stack di un elemento (quale elemento deve essere posizionato di fronte a, o dietro, gli altri). https://www.w3schools.com/css/tryit.asp?filename=trycss_zindex

Overflow - https://www.w3schools.com/css/css_overflow.asp

La ***overflow*** CSS controlla cosa succede ai contenuti che sono troppo grandi per adattarsi a un'area.

La ***overflow*** specifica se ritagliare il contenuto o aggiungere barre di scorrimento quando il contenuto di un elemento è troppo grande per adattarsi all'area specificata:

- ***visible***- Predefinito. L'overflow non è troncato. Il contenuto viene visualizzato all'esterno della casella dell'elemento.
https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_visible
- ***hidden*** - L'overflow è troncato e il resto del contenuto sarà invisibile.
https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_hidden
- ***scroll*** - L'overflow viene tagliato e viene aggiunta una barra di scorrimento per vedere il resto del contenuto. Nota che questo aggiungerà una barra di scorrimento sia in orizzontale che in verticale.
https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_scroll
- ***auto***- Simile a scroll, ma aggiunge barre di scorrimento solo quando necessario.
https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_auto

Le proprietà ***overflow-x*** e ***overflow-y*** specifica se modificare l'overflow del contenuto solo orizzontalmente o verticalmente (o entrambi):

- ***overflow-x***, specifica cosa fare con i bordi sinistro / destro del contenuto.
- ***overflow-y***, specifica cosa fare con i bordi superiore / inferiore del contenuto.

https://www.w3schools.com/css/tryit.asp?filename=trycss_overflow_xy

Float - https://www.w3schools.com/css/css_float.asp

La ***float*** viene utilizzata per il posizionamento e la formattazione del contenuto, ad esempio lascia che un'immagine rimanga nel testo in un contenitore:

- ***left*** - L'elemento fluttua a sinistra del suo contenitore
https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float2
- ***right*** - L'elemento fluttua a destra del suo contenitore
https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float
- ***none*** - L'elemento non galleggia (verrà visualizzato esattamente dove si trova nel testo). Predefinito.
https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_float_none
- ***inherit***: l'elemento eredita il valore float del suo genitore

Nel suo uso più semplice, la ***float*** può essere utilizzata per avvolgere il testo attorno alle immagini.

Clear

La ***clear*** specifica quali elementi possono fluttuare accanto all'elemento cancellato e su quale lato:

- ***none*** - Consente elementi mobili su entrambi i lati. Questo è il valore predefinito
- ***left*** - Nessun elemento flottante consentito sul lato sinistro
- ***right***: non sono ammessi elementi mobili sul lato destro
- ***both(entrambi)*** - Nessun elemento flottante consentito sul lato sinistro o destro
- ***inherit***: l'elemento eredita il valore chiaro del suo genitore

Il modo più comune di usare la ***clear*** è dopo aver usato una ***float*** su un elemento.

https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clear

Clearfix Hack

Se un elemento è più alto dell'elemento che lo contiene ed è ***float***, sarà "overflow" al di fuori del suo contenitore. Quindi possiamo aggiungere ***overflow: auto;*** all'elemento contenitore per risolvere questo problema.

https://www.w3schools.com/css/tryit.asp?filename=trycss_layout_clearfix

Griglia di scatole / scatole di uguale larghezza

Con la ***float***, è facile far galleggiare parallelamente scatole di contenuti.

https://www.w3schools.com/css/tryit.asp?filename=trycss_float_boxes

Puoi facilmente creare tre scatole fluttuanti affiancate. Tuttavia, quando aggiungi qualcosa che ingrandisce la larghezza di ciascuna casella (ad es. Padding o bordi), la finestra si interromperà. La ***box-sizing*** ci consente di includere l'imbottitura e il bordo nella larghezza (e altezza) della scatola, assicurandoti che l'imbottitura rimanga all'interno della scatola e che non si rompa.

https://www.w3schools.com/css/tryit.asp?filename=trycss_float_images_side

Non è facile creare scatole mobili con uguali altezze. Una soluzione rapida è impostare un'altezza fissa.

https://www.w3schools.com/css/tryit.asp?filename=trycss_float_boxes_height

Può allungare automaticamente le scatole per essere lunghe quanto la scatola più lunga.

https://www.w3schools.com/css/tryit.asp?filename=trycss_float_boxes_flex

Menu di navigazione

Utilizzare ***float*** con un elenco di collegamenti ipertestuali per creare un menu orizzontale

https://www.w3schools.com/css/tryit.asp?filename=trycss_float5

Allineamento orizzontale e verticale - https://www.w3schools.com/css/css_align.asp

Centrare orizzontalmente un ***elemento*** del blocco (come <div>), usare ***margin: auto;***

L'impostazione della larghezza dell'elemento impedisce che si estenda fino ai bordi del suo contenitore.

L'elemento prenderà quindi la larghezza specificata e lo spazio rimanente sarà diviso equamente tra i due margini.

https://www.w3schools.com/css/tryit.asp?filename=trycss_align_container

---l'allineamento al centro non ha alcun effetto se la width non è impostata.---

Per centrare il testo all'interno di un elemento, usa ***text-align: center;***

https://www.w3schools.com/css/tryit.asp?filename=trycss_align_text

Per centrare un'immagine, imposta il margine sinistro e destro su ***auto*** e rendilo in un ***block***.

https://www.w3schools.com/css/tryit.asp?filename=trycss_align_image

Un metodo per allineare gli elementi è usare ***position: absolute;***

https://www.w3schools.com/css/tryit.asp?filename=trycss_align_pos

Un altro metodo per allineare gli elementi consiste nell'utilizzare la ***float***.

https://www.w3schools.com/css/tryit.asp?filename=trycss_align_float

Ci sono molti modi per centrare un elemento verticalmente in CSS. Una soluzione semplice è quella di usare in alto e in basso ***padding***. Per centrare sia verticalmente che orizzontalmente, usa ***padding* e *text-align: center***.

https://www.w3schools.com/css/tryit.asp?filename=trycss_align_padding2

Barra di navigazione - https://www.w3schools.com/css/css_navbar.asp

Una barra di navigazione è fondamentalmente una lista di collegamenti, quindi usare gli elementi e ha perfettamente senso.

- ***list-style-type: none;*** - Rimuove i proiettili. Una barra di navigazione non ha bisogno di indicatori di lista
- Impostare ***margin: 0;*** e ***padding: 0;*** rimuovere le impostazioni predefinite del browser
- ***display: block;*** - La visualizzazione dei collegamenti come elementi di blocco rende cliccabile l'intera area di collegamento (non solo il testo), e ci consente di specificare la larghezza (e il riempimento, il margine, l'altezza, ecc)
- ***width: 60px;*** - Gli elementi di blocco occupano l'intera larghezza disponibile per impostazione predefinita. Vogliamo specificare una larghezza di 60 pixel

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_vertical2

Con un colore di sfondo grigio e modifica il colore di sfondo dei collegamenti quando l'utente sposta il mouse su di essi.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_vertical_gray

Aggiungi una classe "***active***" al link corrente per consentire all'utente di sapere su quale pagina si trova.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_vertical_active

Aggiungi ***text-align:center*** a o <a> per centrare i collegamenti.

Aggiungi la ***border*** a aggiungi un bordo attorno alla barra di navigazione. Se vuoi anche i bordi all'interno della barra di navigazione, aggiungi ***border-bottom*** a tutti gli elementi , ad eccezione dell'ultima.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_vertical_borders

Crea una *navigazione laterale* "appiccicosa" a tutta altezza.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_vertical_fixed

Esistono due modi per creare una barra di navigazione orizzontale. Utilizzo di elementi di elenchi in ***linea*** o ***mobili***.

Un modo per costruire una barra di navigazione orizzontale è specificare gli elementi come inline.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_horizontal

display: inline; - Per impostazione predefinita, gli elementi sono elementi di blocco. Qui, rimuoviamo le interruzioni di riga prima e dopo ciascuna voce di elenco.

Crea una barra di navigazione orizzontale di base con un colore di sfondo scuro e modifica il colore di sfondo dei collegamenti quando l'utente sposta il mouse su di essi.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_horizontal_black

Aggiungi una classe "active" al link corrente per consentire all'utente di sapere su quale pagina si trova

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_horizontal_black_active

Rendi la barra di navigazione in cima o in fondo alla pagina, anche quando l'utente scorre la pagina

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_horizontal_black_fixed

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_horizontal_black_fixed2

Utilizzare **position: sticky;** per creare una barra di navigazione adesiva.

https://www.w3schools.com/css/tryit.asp?filename=trycss_navbar_sticky

Dropdown - https://www.w3schools.com/css/css_dropdowns.asp

Crea una casella a discesa che appare quando l'utente sposta il mouse su un elemento.

https://www.w3schools.com/css/tryit.asp?filename=trycss_dropdown_text

La **.dropdown** classe usa **position: relative**, che è necessaria quando vogliamo che il contenuto del menu a discesa venga posizionato proprio sotto il pulsante a discesa (usando **position: absolute**).

La **.dropdown-content** classe contiene il contenuto effettivo a discesa. È nascosto per impostazione predefinita e verrà visualizzato al passaggio del mouse (vedere sotto). Notare che **min-width** è impostato su 160px. Sentiti libero di cambiare questo. **Suggerimento:** se si desidera che la larghezza del contenuto del menu a discesa sia ampia quanto il pulsante a discesa, impostare **width** su 100% (e **overflow: auto** abilitare lo scorrimento su schermi piccoli).

Invece di usare un bordo, abbiamo usato la **box-shadow** proprietà CSS per rendere il menu a discesa simile a una "carta".

Il **:hover** selettore viene utilizzato per mostrare il menu a discesa quando l'utente sposta il mouse sul pulsante a discesa.

https://www.w3schools.com/css/tryit.asp?filename=trycss_dropdown_button

Image Sprites - https://www.w3schools.com/css/css_image_sprites.asp

L'uso degli **sprite** delle immagini riduce il numero di richieste del server e risparmia la larghezza di banda.

Invece di usare tre immagini separate, usiamo questa singola immagine ("img_navsprites.gif"):



Con i CSS, possiamo mostrare solo la parte dell'immagine di cui abbiamo bisogno.

https://www.w3schools.com/css/tryit.asp?filename=trycss_sprites_img

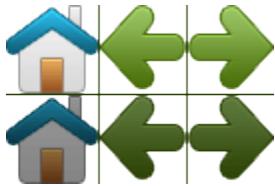
- ****- Definisce solo una piccola immagine trasparente perché l'attributo src non può essere vuoto. L'immagine visualizzata sarà l'immagine di sfondo che specificheremo in CSS
- **width: 46px; height: 44px;** - Definisce la parte dell'immagine che vogliamo usare
- **background: url(img_navsprites.gif) 0 0;** - Definisce l'immagine di sfondo e la sua posizione (0px a sinistra, 0px in alto)

Vogliamo utilizzare l'immagine **sprite** ("img_navsprites.gif") per creare una lista di navigazione.

https://www.w3schools.com/css/tryit.asp?filename=trycss_sprites_nav

Ora vogliamo aggiungere un effetto **hover** alla nostra lista di navigazione.

La nostra nuova immagine ("img_navsprites_hover.gif") contiene tre immagini di navigazione e tre immagini da utilizzare per gli effetti al passaggio del mouse:



Poiché si tratta di una singola immagine e non di sei file separati, non vi sarà **alcun ritardo di caricamento** quando un utente si posiziona sull'immagine.

https://www.w3schools.com/css/tryit.asp?filename=trycss_sprites_hover_nav

Il **[attribute]** selettore viene utilizzato per selezionare elementi con un attributo specificato.

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_attribute

Il **[attribute="value"]** selettore viene utilizzato per selezionare elementi con un attributo e un valore specificati.

L'esempio seguente seleziona tutti gli elementi <a> con un attributo target = "_blank":

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_attribute_value

Il **[attribute~="value"]** selettore viene utilizzato per selezionare elementi con un valore di attributo contenente una parola specificata.

L'esempio seguente seleziona tutti gli elementi con un attributo title che contiene un elenco di parole separate da spazi, una delle quali è "flower":

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_attribute_value2

Il **[attribute|= "value"]** selettore viene utilizzato per selezionare elementi con l'attributo specificato che inizia con il valore specificato.

L'esempio seguente seleziona tutti gli elementi con un valore di attributo di classe che inizia con "top":

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_attribute_hyphen

Domanda 23 (points: from 0 to 4) Scrivi le regole CSS per mostrare il div con:

- bordo in alto, in basso e a destra: rosso, a puntini (dotted), spessore 1px;
- bordo a sinistra: rosso, solid, spessore 5px;
- sfondo giallo, testo centrato e padding di 18px.

```
<!DOCTYPE html>
<html>
<head>
<style>

div {
    border: 1px dotted red;
    border-left: 5px solid red;
    padding: 18px;
    text-align: center;
    background-color: yellow;
}

</style>
</head>
<body>
<div>A specific div</div>
</body>
</html>
```

Bootstrap - https://www.w3schools.com/bootstrap/bootstrap_get_started.asp

Bootstrap è un framework front-end gratuito per uno sviluppo web più rapido e semplice.

Bootstrap include modelli di progettazione basati su HTML e CSS per tipografia, moduli, pulsanti, tabelle, navigazione, modali, caroselli di immagini e molti altri, nonché plug-in JavaScript facoltativi.

Bootstrap ti dà anche la possibilità di creare facilmente modelli reattivi.

Vantaggi del Bootstrap:

- **Facile da usare:** chiunque abbia una conoscenza di base di HTML e CSS può iniziare a utilizzare Bootstrap
- **Caratteristiche Responsive:** reattivo CSS di Sputafuoco si adatta ai telefoni, tablet e desktop
- **Approccio mobile-first:** in Bootstrap 3, gli stili mobile-first fanno parte del core framework
- **Compatibilità con il browser:** Bootstrap è compatibile con tutti i browser moderni

Se non si desidera scaricare e ospitare da soli Bootstrap, è possibile includerlo da un CDN (Content Delivery Network).

MaxCDN fornisce supporto CDN per CSS e JavaScript di Bootstrap:

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
<!-- jQuery library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<!-- Latest compiled JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
```

Crea la prima pagina Web con Bootstrap

1. Aggiungi il doctype HTML5:

Bootstrap utilizza elementi HTML e proprietà CSS che richiedono il doctype HTML5.

Includere sempre il doctype HTML5 all'inizio della pagina, insieme all'attributo lang e al set di caratteri corretto:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
  </head>
</html>
```

2. Bootstrap 3 è mobile-first:

Bootstrap 3 è progettato per essere sensibile ai dispositivi mobili. Gli stili mobile-first fanno parte del framework principale.

Per garantire il corretto rendering e il tocco dello zoom, aggiungi il seguente <meta>tag all'interno

<head> dell'elemento:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

La parte **width=device-width** imposta la larghezza della pagina in modo che segua la larghezza dello schermo del dispositivo (che varierà a seconda del dispositivo).

La parte **initial-scale=1** imposta il livello di zoom iniziale quando la pagina viene caricata per la prima volta dal browser.

3. Contenitori:

Bootstrap richiede anche un elemento contenente per avvolgere il contenuto del sito.

Esistono due classi di contenitori tra cui scegliere:

- La classe **.container** fornisce un contenitore a larghezza fissa reattiva
https://www.w3schools.com/bootstrap/tryit.asp?filename=trybs_gs_container&stacked=h
- La classe **.container-fluid** fornisce un contenitore a larghezza intera , che copre l'intera larghezza del viewport
https://www.w3schools.com/bootstrap/tryit.asp?filename=trybs_gs_container-fluid&stacked=h

.contenitore

.container-fluid

Grid - https://www.w3schools.com/bootstrap/bootstrap_grid_basic.asp

Il sistema di griglia di Bootstrap consente fino a 12 colonne nella pagina.

Se non si desidera utilizzare tutte e 12 le colonne singolarmente, è possibile raggruppare le colonne per creare colonne più larghe:

intervallo 1												
intervallo 4				intervallo 4				intervallo 4				
intervallo 4				span 8				span 8				
intervallo 6						intervallo 6						span 12
span 12						span 12						span 12

Il sistema di griglia Bootstrap è reattivo e le colonne si riorganizzano automaticamente in base alle dimensioni dello schermo.

Il sistema di griglia Bootstrap ha quattro classi:

- **xs** (per telefoni - schermi con larghezza inferiore a 768 px)
- **sm** (per tablet - schermi pari o superiori a 768 px di larghezza)
- **md** (per portatili piccoli - schermi pari o superiori a 992 px di larghezza)
- **lg** (per laptop e desktop - schermi pari o superiori a 1200px di larghezza)

Le classi sopra possono essere combinate per creare layout più dinamici e flessibili

Struttura di base di una griglia di bootstrap

```
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>
<div class="row">
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
  <div class="col-*-*"></div>
</div>
<div class="row">
  ...
</div>
```

Primo; crea una riga (**<div class="row">**). Quindi, aggiungi il numero desiderato di colonne (tag con **.col-*-*** classi appropriate). Nota che i numeri in **.col-*-*** devono sempre aggiungere fino a 12 per ogni riga.

https://www.w3schools.com/bootstrap/tryit.asp?filename=trybs_grid_ex1&stacked=h

XML (eXtensible Markup Language) - https://www.w3schools.com/xml/xml_whatis.asp

È uno strumento indipendente dal software e dall'hardware per la memorizzazione e il trasporto dei dati.

L'XML è abbastanza **auto-descrittivo**:

- Ha informazioni sul mittente.
- Ha informazioni sul ricevitore
- Ha un titolo
- Ha un corpo di messaggio.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML e HTML sono stati progettati con diversi obiettivi:

- XML è stato progettato per il trasporto di dati, con particolare attenzione ai dati
- L'HTML è stato progettato per visualizzare i dati, con particolare attenzione all'aspetto dei dati
- I tag XML non sono predefiniti come i tag HTML

Con XML, l'autore deve definire sia i tag che la struttura del documento.

La maggior parte delle applicazioni XML funzionerà anche se vengono aggiunti (o rimossi) nuovi dati.

XML separa i dati dalla presentazione, non contiene alcuna informazione su come vengono visualizzati, e possono essere utilizzati in molti diversi scenari di presentazione.

XML separa i dati da HTML, quando si visualizzano i dati in HTML, non è necessario modificare il file HTML quando i dati cambiano. Con XML, i dati possono essere archiviati in file XML separati.

*Un documento XML si dice **ben formato (well-formed)** se sono rispettate le seguenti regole:*

- Il documento XML contiene un unico elemento **root** ("radice" dell'albero);
- Gli elementi devono essere sempre chiusi con **tag di chiusura** o, se vuoti, tramite chiusura abbreviata (/>);
- Bisogna rispettare l'**ordine di nidificazione**: un elemento padre non può essere chiuso prima di un elemento figli;
- XML è **case sensitive**: bisogna ricordarlo quando usiamo maiuscole e minuscole per nomi dei tag e attributi
- Gli **attributi** devono essere racchiusi tra singoli o doppi apici

Regole di sintassi - https://www.w3schools.com/xml/xml_syntax.asp

- I documenti XML devono contenere un elemento **radice** che è il **genitore** di tutti gli altri elementi
- Il **prologo** XML è facoltativo. Se esiste, deve venire prima nel documento. Per evitare errori, è necessario specificare la codifica utilizzata o salvare i file XML come UTF-8 (codifica dei caratteri predefinita per i documenti XML).
- Tutti gli elementi **devono** avere un tag di chiusura.
- I tag XML sono **case sensitive**. Il tag <Lettera> è diverso dal tag <lettera>.
- Gli elementi XML devono essere **nidificati correttamente**.
- Gli elementi XML possono avere **attributi** in coppie nome / valore proprio come in HTML, e i suoi valori devono sempre essere citati.
- **Riferimenti di entità**, es. sostituire il carattere "<" con un **riferimento di entità**.

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

- La sintassi per scrivere **commenti** in XML è simile a quella dell'HTML: <!-- This is a comment -->
- **XML non tronca più spazi bianchi**.

Si dice che i documenti XML conformi alle regole di sintassi sopra siano documenti XML "***ben formati***".

Albero XML - https://www.w3schools.com/xml/xml_tree.asp

I documenti XML sono formati come ***alberi di elementi***. Un albero XML inizia da un elemento radice coi suoi elementi figlio. Tutti gli elementi possono avere elementi secondari.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML utilizza una sintassi molto auto-descrittiva:

- Un ***prolog*** definisce la versione XML e la codifica dei caratteri:

```
<?xml version="1.0" encoding="UTF-8"?>
```

- La riga successiva è l'***elemento principale*** del documento:

```
<bookstore>
```

- La riga successiva inizia un elemento ***<book>***:

```
<book category="cooking">
```

- Gli elementi ***<book>*** hanno 4 elementi figlio : ***<title>***, ***<author>***, ***<year>***, ***<price>***:

```
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
```

Elementi - https://www.w3schools.com/xml/xml_elements.asp

Un elemento XML è tutto da (incluso) il tag di inizio dell'elemento a (incluso) il tag di fine.

<price>29.99</price>

Un elemento può contenere:

- testo
- attributi
- altri elementi
- un mix di quanto sopra

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Elementi XML vuoti, si dice che un elemento senza contenuto sia vuoto.

<element></element> = <element /> (tag di chiusura automatica)

Gli elementi vuoti possono avere attributi.

Gli elementi XML devono seguire queste **regole di denominazione**:

- sono sensibili al maiuscolo / minuscolo
- devono iniziare con una lettera o un trattino basso
- non possono iniziare con le lettere xml (o XML, o Xml, ecc.)
- possono contenere lettere, cifre, trattini, underscore e punti
- non possono contenere spazi

Come questo: <book_title> e non come questo: <the_title_of_the_book>.

Attributi - https://www.w3schools.com/xml/xml_attributes.asp

Gli attributi sono progettati per contenere dati relativi a un elemento specifico.

I valori degli attributi devono sempre essere citati. È possibile utilizzare virgolette singole o doppie.

<person gender="female"> oppure <person gender='female'>

Alcune cose da considerare quando si usano gli attributi sono:

- non possono contenere più valori (elementi possono)
- non possono contenere strutture ad albero (elementi possono)
- non sono facilmente espandibili (per modifiche future)

A volte i riferimenti ID sono assegnati agli elementi. Questi ID possono essere utilizzati per identificare gli elementi XML più o meno allo stesso modo dell'attributo id in HTML.

Gli attributi id sono per identificare le diverse note. Non fa parte della nota stessa.

I metadati (dati sui dati) dovrebbero essere memorizzati come attributi.

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me!</body>
  </note>
</messages>
```

Namespace - https://www.w3schools.com/xml/xml_namespaces.asp

I **namespace** XML forniscono un metodo per evitare conflitti tra i nomi degli elementi.

In XML, i nomi degli elementi sono definiti dallo sviluppatore. Ciò si traduce spesso in un conflitto quando si tenta di combinare documenti XML da diverse applicazioni XML.

Risolvere il conflitto del nome usando un prefisso:

I conflitti dei nomi possono essere facilmente evitati usando un prefisso del nome: **h:table**

L'attributo xmlns:

Quando si utilizzano prefissi in XML, deve essere definito uno **spazio** dei **nomi** per il prefisso che può essere definito da un attributo **xmlns** nel tag di inizio di un elemento: **xmlns: prefisso="URI"**.

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="https://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

Quando uno spazio dei nomi è definito per un elemento, tutti gli elementi figlio con lo stesso prefisso sono associati allo stesso spazio dei nomi.

Lo scopo dell'utilizzo di un URI è di dare allo spazio dei nomi un nome univoco.

I **namespace** possono anche essere dichiarati nell'elemento **radice** XML.

HttpRequest - https://www.w3schools.com/xml/xml_http.asp

Tutti i browser moderni dispongono di un oggetto **XMLHttpRequest** integrato per richiedere dati da un server.

L'oggetto **XMLHttpRequest** è un sogno per gli sviluppatori, perché puoi:

- Aggiorna una pagina Web senza ricaricare la pagina
- Richiedi dati da un server - dopo che la pagina è stata caricata
- Ricevi dati da un server - dopo che la pagina è stata caricata
- Invia dati a un server - in background

https://www.w3schools.com/xml/tryit.asp?filename=tryxml_httprequest

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    // Typical action to be performed when the document is ready:
    document.getElementById("demo").innerHTML = xhttp.responseText;
  }
};
xhttp.open("GET", "filename", true);
xhttp.send();
```

1. La prima riga nell'esempio sopra crea un oggetto **XMLHttpRequest**:

```
var xhttp = new XMLHttpRequest();
```

- La proprietà **onreadystatechange** specifica una funzione da eseguire ogni volta che cambia lo stato dell'oggetto XMLHttpRequest: `xhttp.onreadystatechange = function()`
- Quando la proprietà **readyState** è 4 e la proprietà **status** è 200, la risposta è pronta:
`if (this.readyState == 4 && this.status == 200)`
- La proprietà **responseText** restituisce la risposta del server come una stringa di testo. La stringa di testo può essere utilizzata per aggiornare una pagina Web:
`document.getElementById("demo").innerHTML = xhttp.responseText;`

Parser - https://www.w3schools.com/xml/xml_parser.asp

Il **DOM XML (Document Object Model)** definisce le proprietà e i metodi per accedere e modificare XML.

Tuttavia, prima di poter accedere a un documento XML, è necessario caricarlo in un oggetto DOM XML.

Tutti i browser moderni hanno un parser XML integrato che può convertire il testo in un oggetto DOM XML.

L' **oggetto XMLHttpRequest** ha un parser XML incorporato.

La proprietà **responseText** restituisce la risposta come una stringa.

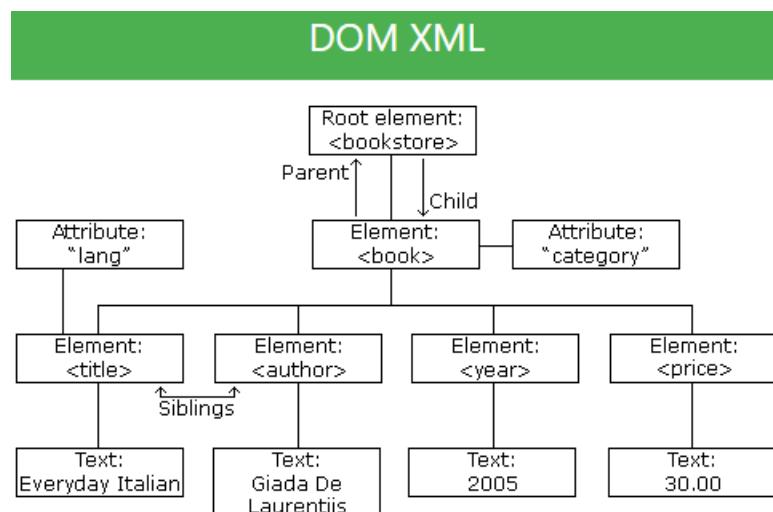
La proprietà **responseXML** restituisce la risposta come un oggetto DOM XML.

Se si desidera utilizzare la risposta come oggetto DOM XML, è possibile utilizzare la proprietà **responseXML**.

https://www.w3schools.com/xml/tryit.asp?filename=try_dom_xmlhttprequest_responsetext

DOM - https://www.w3schools.com/xml/xml_dom.asp

Il DOM XML definisce un modo standard per accedere e manipolare i documenti XML. Presenta un documento XML come *struttura ad albero*.



È possibile accedere a tutti gli elementi XML tramite il DOM XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Questo codice recupera il valore di testo del primo elemento `<title>` in un documento XML:

```
txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

Il DOM XML è uno standard su come ottenere, modificare, aggiungere ed eliminare elementi XML.

DTD (Document Type Definition) - https://www.w3schools.com/xml/xml_dtd_intro.asp

Un documento XML "**valido**" è un documento XML "**ben formato**", che è anche conforme alle regole **DTD**.

- Definisce la struttura di ogni elemento, non si possono usare altri elementi se non quelli definiti.
- Dichiara una serie di attributi per ogni elemento e che valori possono o devono assumere
- Fornisce meccanismi per semplificare la gestione del documento (entity) e importare parti di altri DTD.

Dichiarazione interna:

Se il DTD è dichiarato all'interno del file XML, deve essere racchiuso all'interno della definizione **<!DOCTYPE>**

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

https://www.w3schools.com/xml/note_in_dtd.xml

- **!DOCTYPE note** definisce che l'elemento radice di questo documento è note
- **!ELEMENT note** definisce che l'elemento note deve contenere quattro elementi: "to, from, heading, body"
- **!ELEMENT to** definire l'elemento to di tipo "#PCDATA"
- **!ELEMENT from** definisce l'elemento from di tipo "#PCDATA"
- **!ELEMENT heading** definisce l'elemento di intestazione di tipo "#PCDATA"
- **!ELEMENT body** definisce l'elemento body come di tipo "#PCDATA"

Dichiarazione esterna:

Se la **DTD** è dichiarata in un file esterno, la definizione **<!DOCTYPE>** deve contenere un riferimento al file.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

File "note.dtd":

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML Building Blocks - https://www.w3schools.com/xml/xml_dtd_building.asp

Visto da un punto di vista DTD, tutti i documenti XML sono costituiti dai seguenti elementi:

- **Elementi:** sono i componenti principali di entrambi i documenti XML ("note" e "messaggi") e HTML("body" e "table"). Gli elementi possono contenere testo, altri elementi o essere vuoti.
- **Attributi:** forniscono ulteriori informazioni sugli elementi, sono sempre collocati all'interno del tag di apertura di un elemento (nome / valore).
Es: ``.
Il nome dell'elemento è "img". Il nome dell'attributo è "src". Il valore dell'attributo è "computer.gif". Poiché l'elemento stesso è vuoto, viene chiuso da un "/".
- **Entità:** Alcuni caratteri hanno un significato speciale in XML, come il segno di minore (<) che definisce l'inizio di un tag XML.

Entity References	Character
<	<
>	>
&	&
"	"
'	'

- **PCDATA(parsed character data):** significa dati di carattere analizzati. Ovvero, un testo che verrà analizzato da un **parser**. Il testo sarà esaminato dal **parser** per entità e markup. I tag all'interno del testo verranno trattati come markup e le entità verranno espanso.
- **CDATA(character data):** un testo che **NON** verrà analizzato da un **parser**. I tag all'interno del testo NON verranno trattati come markup e le entità non verranno espanso.

Elementi - https://www.w3schools.com/xml/xml_dtd_elements.asp

In un DTD, gli elementi XML sono dichiarati con la seguente sintassi:

`<!ELEMENT element-name category>` or `<!ELEMENT element-name (element-content)>`

Gli elementi possono avere parole chiavi come:

- **EMPTY, elementi vuoti:** `<!ELEMENT element-name EMPTY>`
Example: `<!ELEMENT br EMPTY>`
XML example: `
`
- **#PCDATA, tra parentesi:** `<!ELEMENT element-name (#PCDATA)>`
Example: `<!ELEMENT from (#PCDATA)>`
- **ANY, combinazione di dati analizzabili:** `<!ELEMENT element-name ANY>`
Example: `<!ELEMENT note ANY>`
- **Sequenze di figli:** `<!ELEMENT element-name (child1,child2..)>`
Example: `<!ELEMENT note (to,from,heading,body)>`

Quando i bambini vengono dichiarati in una sequenza separata da virgole, i bambini devono apparire nella stessa sequenza nel documento. La dichiarazione completa dell'elemento "note" è:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

- **Dichiarazione di minima occorrenza di un elemento "+",** il figlio deve essere presente una o più volte:

```
<!ELEMENT element-name (child-name+)>
Example:      <!ELEMENT note (message+)>
```

- **Dichiarare zero o più occorrenze di un elemento "**",** il figlio può essere presente zero o più volte:

```
<!ELEMENT element-name (child-name*)>
Example:      <!ELEMENT note (message*)>
```

- **Dichiarare zero o una occorrenza di un elemento "?",** il figlio può essere presente zero o una volta:

```
<!ELEMENT element-name (child-name?)>
Example:      <!ELEMENT note (message?)>
```

- **Dichiarazione contenuto OR "/**, "note" deve contenere un elemento "to" e un "message" o un elemento "body".

```
<!ELEMENT note (to, (message|body))>
```

- **Dichiarazione di contenuti misti,** "note" può contenere zero o più occorrenze di dati di carattere analizzati, elementi "to" o " message ".

```
<!ELEMENT note (#PCDATA|to|message)*>
```

Attributi - https://www.w3schools.com/xml/xml_dtd_attributes.asp

Una dichiarazione di attributo ha la seguente sintassi:

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

DTD example: <!ATTLIST payment type CDATA "check">

Gli attributi possono avere parole chiavi come:

- **Attributo predefinito,** l'elemento "square" è definito come un elemento vuoto con un attributo "width" di tipo CDATA. Se non viene specificata alcuna larghezza, ha un valore predefinito di "0".

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

- **#REQUIRED,** se non hai un'opzione per un valore predefinito, ma vuoi comunque forzare la presenza dell'attributo.

```
<!ATTLIST person number CDATA #REQUIRED>
```

- **#IMPLIED,** se non vuoi forzare l'autore a includere un attributo e non hai un'opzione per un valore predefinito.

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

- **#FIXED,** quando vuoi che un attributo abbia un valore fisso senza consentire all'autore di cambiarlo. Se un autore include un altro valore, il parser XML restituirà un errore.

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

- **Valori attributo numerati,** quando gli attributi devono assumere un valore di un insieme.

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
<!ATTLIST payment type (check|cash) "cash">
```

Elementi vs Attributi - https://www.w3schools.com/xml/xml_dtd_el_vs_attr.asp

I dati possono essere memorizzati in elementi figlio o in attributi.

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

Entrambi gli esempi forniscono le stesse informazioni. Gli attributi sono utili in HTML, ma in XML si dovrebbe cercare di evitarli. Usa elementi figli se le informazioni sembrano dati.

Alcuni dei problemi con gli attributi sono:

- non possono contenere più valori (elementi figlio sì)
- non sono facilmente espandibili (per modifiche future)
- non possono descrivere strutture (elementi figlio sì)
- sono più difficili da manipolare in base al codice del programma
- i valori degli attributi non sono facili da testare con un DTD

A volte assegno i riferimenti ID possono essere utilizzati

per accedere agli elementi XML più o meno allo stesso modo degli attributi NAME o ID in HTML, per identificare le diverse note nel file XML e non una parte dei dati della nota. I metadati (dati sui dati) dovrebbero essere memorizzati come attributi e che i dati stessi dovrebbero essere archiviati come elementi.

```
<note>
  <date>
    <day>12</day>
    <month>11</month>
    <year>2002</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
<messages>
<note id="p501">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

<note id="p502">
  <to>Jani</to>
  <from>Tove</from>
  <heading>Re: Reminder</heading>
  <body>I will not!</body>
</note>
</messages>
```

Entità - https://www.w3schools.com/xml/xml_dtd_entities.asp

Le entità vengono utilizzate per definire scorciatoie per caratteri speciali, possono essere dichiarate interne o esterne.

Dichiarazione entità interna:

```
<!ENTITY entity-name "entity-value">
```

DTD Example:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```

Dichiarazione entità esterna:

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

DTD Example:

```
<!ENTITY writer SYSTEM "https://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM "https://www.w3schools.com/entities.dtd">
```

JavaScript

JavaScript è un linguaggio di scripting sviluppato per dare interattività alle pagine HTML, può essere inserito direttamente nelle pagine Web. *Javascript è considerato un linguaggio di scripting di tipo interpretato e client-side.*

JavaScript è :

- **interpretato** e non compilato
- **object-based** ma non **class-based**, esiste il concetto di oggetto ma non di classe
- **debolmente tipizzato** (weakly typed), non è necessario definire il tipo di una variabile

Il codice JavaScript viene eseguito da un interprete contenuto all'interno del browser

Nasce per dare **dinamicità** alle pagine Web, consente di:

- Accedere e modificare elementi della pagina HTML
- Reagire ad eventi generati dall'interazione fra utente e pagina
- Validare i dati inseriti dall'utente
- determinare il browser utilizzato e la dimensione della finestra in cui viene mostrata la pagina, lavorare con i browser cookie, ecc...

Sintassi del linguaggio

La sintassi di JavaScript è modellata su quella del C con alcune varianti significative.

- È un linguaggio **case-sensitive**
- Le istruzioni sono terminate da ';' ma il terminatore può essere omesso se si va a capo
- Sono ammessi sia commenti multilinea (delimitati da /* e */) che monolinea (iniziano con //)
- Gli identificatori possono contenere lettere, cifre e i caratteri '_' e '\$', ma non possono iniziare con una cifra
- La sintassi JavaScript definisce due tipi di valori: valori fissi e valori variabili:
 - I valori fissi sono chiamati **letterali**
 - I valori variabili sono chiamati **variabili**

Posizionamento di codice JavaScript - https://www.w3schools.com/js/js_whereto.asp

In HTML, il codice JavaScript deve essere inserito tra <script> e </script> tag.

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

Un JavaScript **function** è un blocco di codice JavaScript, che può essere eseguito quando lo "chiamato".

Ad esempio, una funzione può essere chiamata *quando si verifica un evento*, come quando l'utente fa clic su un pulsante.

Gli script possono essere inseriti nella <body>, o nella <head> sezione di una pagina HTML, o in entrambi.

Il posizionamento degli script nella parte inferiore dell'elemento <body> migliora la velocità di visualizzazione, poiché l'interpretazione degli script rallenta la visualizzazione.

Gli script possono essere inseriti anche in file esterni:

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

Per utilizzare uno script esterno, inserisci il nome del file di script nell'attributo **src** (origine) di un <script> tag:

```
<script src="myScript.js"></script>
```

Gli script esterni non possono contenere <script> tag.

Output - https://www.w3schools.com/js/js_output.asp

JavaScript può "visualizzare" i dati in diversi modi:

- Scrivere in un elemento HTML, usando **innerHTML**
- Scrivere nell'output HTML utilizzando **document.write()**
- Scrivere in una casella di avviso, utilizzando **window.alert()**
- Scrivere nella console del browser, utilizzando **console.log()**

innerHTML:

Per accedere a un elemento HTML, JavaScript può utilizzare il metodo **document.getElementById(id)**.

L'attributo **id** definisce l'elemento HTML. La **innerHTML** proprietà definisce il contenuto HTML:

```
<p id="demo"></p>
<script>
    document.getElementById("demo").innerHTML = 5 + 6;
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_output\_dom
```

document.write ():

A scopo di test, è conveniente usare **document.write()**:

```
<script>
    document.write(5 + 6);
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_output\_write
```

L'uso di **document.write ()** dopo il caricamento di un documento HTML, cancellerà tutto il codice HTML esistente :

```
<button type="button" onclick="document.write(5 + 6)">Try it</button>
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_output\_write\_over
```

window.alert ():

È possibile utilizzare una casella di avviso per visualizzare i dati:

```
<script>
    window.alert(5 + 6);
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_output\_alert
```

console.log ():

Per scopi di **debug**, è possibile utilizzare il **console.log()** metodo per visualizzare i dati.

https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_console

Istruzioni - https://www.w3schools.com/js/js_statements.asp

Un **programma JavaScript** è un elenco di **istruzioni** di programmazione .

In HTML, i programmi JavaScript vengono eseguiti dal browser web.

Le istruzioni JavaScript sono composte da: Valori, operatori, espressioni, parole chiave e commenti.

Operatori - https://www.w3schools.com/js/js_operators.asp

Operatori aritmetici sono usati per eseguire aritmetica sui numeri:

+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Operatori di assegnazione assegnano valori alle variabili JavaScript:

=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

Variabili - https://www.w3schools.com/js/js_datatypes.asp

Le variabili vengono dichiarate usando la parola chiave **var**: **var nomevariabile;**

Non hanno un tipo, tecnicamente ha il valore di **undefined**

È prevista la possibilità di inizializzare una variabile contestualmente alla dichiarazione: **var f = 15.8**

Esiste lo **scope globale** e quello **locale** (ovvero dentro una funzione) ma, a differenza di Java, non esiste lo scope di blocco.

Se inserisci un numero tra virgolette, il resto dei numeri verrà trattato come stringhe e concatenato.

```
var x = "5" + 2 + 3;           ->      523
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_variables\_add\_string\_number

var x = 2 + 3 + "5";          ->      55
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_variables\_add\_number\_string
```

Oggetto - https://www.w3schools.com/js/js_objects.asp

Tutti i valori JavaScript, eccetto quelli **primitivi**, sono oggetti.

Un **valore primitivo** è un valore che non ha proprietà o metodi. Un **tipo di dati primitivi** sono dati che hanno un valore primitivo. JavaScript definisce 5 tipi di dati primitivi:

- string
 - number
 - boolean
 - null
 - undefined

I valori primitivi sono immutabili (non possono essere modificati).

Le ***variabili*** JavaScript possono contenere singoli valori:

```
var person = "John Doe";
```

Anche gli oggetti sono variabili, ma possono contenere molti valori, che sono scritti come coppie **nome:valori**:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Un oggetto JavaScript è una raccolta di **valori denominati**.

Proprietà dell'oggetto - https://www.w3schools.com/js/js_object_properties.asp

I valori nominati, negli oggetti JavaScript, sono chiamati ***proprietà***, sono i valori associati a un oggetto.

Un oggetto JavaScript è una raccolta di proprietà non ordinate. Le proprietà possono in genere essere modificate, aggiunte e cancellate, ma alcune sono di sola lettura.

È possibile accedere alle proprietà dell'oggetto in due modi:

objectName.propertyName *oppure* *objectName["propertyName"]*

```
Es    person.lastName;                                person["lastName"];
```

Gli **oggetti** JavaScript sono scritti con parentesi **graffe{}}, le **proprietà** dell'oggetto sono scritte come **nome:valori**, separate da **virgolette**.**

```
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id        : 5566,  
};
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_object

L'istruzione ***for...in*** (*Loop*) scorre le proprietà di un oggetto. In *loop* attraverso le proprietà di un oggetto:

```
var person = {fname:"John", lname:"Doe", age:25};  
for (x in person) {  
    txt += person[x];  
}  
document.write(txt);
```

È possibile **aggiungere nuove proprietà** a un oggetto esistente semplicemente assegnandogli un valore.

```
person.nationality = "English";
```

La parola chiave **delete** elimina una proprietà da un oggetto, cancella sia il valore della proprietà sia la proprietà stessa:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
delete person.age; // or delete person["age"];
```

In JavaScript, tutti gli **attributi** possono essere letti, ma solo l'attributo **value** può essere modificato.

Metodi dell'oggetto - https://www.w3schools.com/js/js_object_methods.asp

I metodi sono **azioni** che possono essere eseguite sugli oggetti. Le proprietà dell'oggetto possono essere sia valori primitivi, altri oggetti e funzioni.

Un **metodo object** è una proprietà dell'oggetto che contiene una **definizione di funzione**.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_method

Gli oggetti JavaScript sono contenitori per valori denominati, chiamati proprietà e metodi.

In una definizione di funzione, **this** riferisce al "proprietario" della funzione.

Nell'esempio sopra, **this** è l' **oggetto persona** che "possiede" la funzione **fullName**, **this.firstName** indica la proprietà **firstName** di **questo oggetto** .

Si accede a un metodo oggetto con la seguente sintassi: **objectName.methodName()**

Se si accede alla **proprietà** **fullName** , senza (), verrà restituita la **definizione della funzione**:

```
name = person.fullName; https://www.w3schools.com/js/tryit.asp?filename=tryjs\_object\_function
```

Metodi incorporati:

```
var message = "Hello world!";
var x = message.toUpperCase();
```

Il valore di x, dopo l'esecuzione del codice di cui sopra sarà: HELLO WORLD!

Aggiunta di un metodo a un oggetto:

```
person.name = function () {
  return this.firstName + " " + this.lastName;
};
```

-Creazione di un oggetto JavaScript-

Esistono diversi modi per creare nuovi oggetti:

- Definisci e crea un singolo oggetto, usando un **oggetto letterale**.
- Definire e creare un singolo oggetto, con la parola chiave **new**.
- Definire un **costruttore** di oggetti e quindi creare oggetti del tipo costruito.

Oggetto letterale

Usando un oggetto letterale, definisci e crei un oggetto in un'istruzione. Un oggetto letterale è un elenco di coppie **nomi: valori** (come età: 50) all'interno di parentesi **graffe {}**.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Parola chiave new

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Costruttori oggetti

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor

Il modo per creare un "tipo di oggetto" è utilizzare una **funzione di costruzione di oggetti** .

Nell'esempio sopra, **function Person()** è una funzione di costruzione di un oggetto.

Gli oggetti dello stesso tipo vengono creati chiamando la funzione di costruzione con la parola chiave **new**:

```
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

Aggiunta di una proprietà/metodo a un costruttore

Per aggiungere una nuova proprietà/metodo a un costruttore, è necessario aggiungerlo alla funzione di costruzione:

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.nationality = "English";
this.name = function() { return this.firstName + " " + this.lastName; }
}
```

Gli oggetti JavaScript sono mutabili

Gli oggetti sono mutabili: sono indirizzati per **riferimento, non per valore**.

```
var x = person; // This will not create a copy of person.
```

Qualsiasi modifica a x cambierà anche la persona, perché x e persona sono lo stesso oggetto.

Scope - https://www.w3schools.com/js/js_scope.asp

JavaScript ha uno scope di funzioni: ogni funzione crea un nuovo scope.

Lo scope determina l'accessibilità (visibilità) di queste variabili.

Le variabili dichiarate all'interno di una funzione JavaScript, diventano **LOCALI** per le funzioni, sono accessibili solo dalla funzione.

```
// code here can NOT use carName
function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}
```

Poiché le variabili locali sono riconosciute solo all'interno delle loro funzioni, è possibile utilizzare variabili con lo stesso nome in diverse funzioni.

Una variabile dichiarata all'esterno di una funzione diventa **GLOBALE**, tutti gli script e le funzioni su una pagina web possono accedervi.

```
var carName = "Volvo";
// code here can use carName
function myFunction() {
  // code here can also use carName
}
```

Se assegna un valore a una variabile che non è stata dichiarata, diventerà automaticamente una variabile **GLOBAL**.

Questo esempio di codice dichiarerà una variabile globale **carName**, anche se il valore è assegnato all'interno di una funzione.

```
myFunction();
// code here can use carName
function myFunction() {
```

```
carName = "Volvo";
}
```

Con JavaScript, l'ambito globale è l'ambiente JavaScript completo.

In HTML, l'ambito globale è l'oggetto finestra. Tutte le variabili globali appartengono all'oggetto della finestra.

```
var carName = "Volvo";
// code here can use window.carName
```

```
document.getElementById("demo").innerHTML = "I can display " + window.carName;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_scope_window

La durata di una variabile JavaScript inizia quando viene dichiarata.

Le variabili locali vengono cancellate quando la funzione è completata.

In un browser Web, le variabili globali vengono eliminate quando si chiude la finestra del browser (o scheda), ma rimangono disponibili per le nuove pagine caricate nella stessa finestra.

Tipo di operatore

È possibile utilizzare l'**typeof** operatore JavaScript per trovare il tipo di una variabile JavaScript, restituisce il tipo di una variabile o un'espressione:

```
typeof ""           // Returns "string"
typeof "John"       // Returns "string"
typeof 314          // Returns "number"
typeof 3.14         // Returns "number"
typeof (3)          // Returns "number"
```

Qualsiasi variabile può essere svuotata, impostando il valore su **undefined**. Sarà anche il tipo **undefined**.

https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_2

Null

In JavaScript **null** è "niente". Dovrebbe essere qualcosa che non esiste. Sfortunatamente, in JavaScript, il tipo di dati **null** è un oggetto.

Puoi considerarlo un bug in JavaScript che **typeof null** è un oggetto. Dovrebbe essere null.

https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_null

Funzioni - https://www.w3schools.com/js/js_functions.asp

Una funzione JavaScript è un blocco di codice progettato per eseguire una determinata attività.

```
function myFunction(p1, p2) {
  return p1 * p2;      // The function returns the product of p1 and p2
}
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_functions

Una funzione JavaScript è definita con la **function** parola chiave, seguita da un **nome**, seguito da parentesi **()**.

I nomi delle funzioni possono contenere lettere, cifre, trattini bassi e simboli del dollaro (stesse regole delle variabili).

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Gli **argomenti** di funzione sono i **valori** ricevuti dalla funzione quando viene richiamata.

All'interno della funzione, gli argomenti (i parametri) si comportano come variabili locali.

L'accesso a una funzione senza () restituirà la definizione della funzione anziché il risultato della funzione:

```
function toCelsius(fahrenheit) {  
    return (5/9) * (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius_2

Le funzioni possono essere utilizzate nello stesso modo in cui si usano le variabili, in tutti i tipi di formule, assegnazioni e calcoli.

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_function_variable

Si può distinguere fra tipi valore e tipi riferimento

- Numeri e booleani sono tipi valore
- Array e Oggetti sono tipi riferimento

Per le stringhe abbiamo una situazione incerta, pur essendo un tipo primitivo si comportano come un tipo riferimento.

Stringhe - https://www.w3schools.com/js/js_strings.asp

Per trovare la lunghezza di una stringa, usa la **length** proprietà built-in :

```
var txt = "ABCDEFGHIJKLMNPQRSTUVWXYZ";  
var sln = txt.length; https://www.w3schools.com/js/tryit.asp?filename=tryjs\_string\_length
```

Poiché le stringhe devono essere scritte tra virgolette, è utilizzare il carattere di **escape backslash**.

Il \ carattere escape backslash () trasforma i caratteri speciali in caratteri stringa:

```
var x = "We are the so-called \"Vikings\" from the north.";
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_escape_quotes2

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\\	\	Backslash

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

Puoi anche suddividere una riga di codice **all'interno di una stringa di testo** con una singola barra rovesciata:

```
document.getElementById("demo").innerHTML = "Hello \  
Dolly!";  
  
document.getElementById("demo").innerHTML = "Hello " +  
"Dolly!";
```

Normalmente, le stringhe di JavaScript sono valori primitivi, creati da letterali: `var firstName = "John";`

Ma le stringhe possono anche essere definite come oggetti con la parola chiave new:

```
var firstName = new String("John");
```

Quando si utilizza l'operatore `==`, le stringhe uguali sono uguali:

```
var x = "John";  
var y = new String("John");  
// (x == y) is true because x and y have equal values
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object1

Quando si utilizza l'operatore `==`, le stringhe uguali non sono uguali, poiché l'operatore `==` si aspetta l'uguaglianza sia nel tipo che nel valore.

Notare la differenza tra `(x==y)` e `(x==y)`. Il confronto tra due oggetti JavaScript verrà **sempre** restituito `false`.

Metodi di stringa - https://www.w3schools.com/js/js_string_methods.asp

La proprietà **length** restituisce la lunghezza di una stringa:

```
var sln = txt.length;
```

Il metodo **indexOf()** restituisce l'indice dell'occorrenza **first** di un testo specificato in una stringa (Case Sensitive):

```
var pos = str.indexOf("locate");
```

Il metodo **lastIndexOf()** restituisce l'indice dell'**ultima** occorrenza di un testo specificato in una stringa:

```
var pos = str.lastIndexOf("locate");
```

Entrambi **indexOf()** e **lastIndexOf()** restituiscono -1 se il testo non viene trovato.

Entrambi i metodi accettano un secondo parametro come posizione iniziale per la ricerca:

```
var str = "Please locate where 'locate' occurs!";  
var pos = str.indexOf("locate", 15);
```

Il metodo **search()** ricerca una stringa per un valore specificato e restituisce la posizione della corrispondenza:

```
var pos = str.search("locate");
```

I due metodi **NON** sono uguali. Queste sono le differenze:

- Il metodo **search()** non può accettare un secondo argomento di posizione iniziale.
- Il metodo **indexOf()** non può assumere potenti valori di ricerca (espressioni regolari).

Se si omette il secondo parametro, **substring()** verrà troncato il resto della stringa:

```
var str = "Apple, Banana ,Kiwi";  
var res = str.substring(7, 13);           →          Banana
```

Il metodo **replace()** sostituisce un valore specificato con un altro valore in una stringa:

```
str = "Please visit Microsoft!";  
var n = str.replace("Microsoft", "W3Schools");
```

Il **replace()** metodo non modifica la stringa su cui è chiamato. Restituisce una nuova stringa.

Per sostituire la distinzione tra maiuscole e minuscole, utilizzare ***un'espressione regolare*** con una bandiera **/i** (non sensibile):

```
str = "Please visit Microsoft!";
var n = str.replace(/MICROSOFT/i, "W3Schools");
```

Per sostituire tutte le corrispondenze, utilizza ***un'espressione regolare*** con una bandiera **/g** (corrispondenza globale):

```
str = "Please visit Microsoft and Microsoft!";
var n = str.replace(/Microsoft/g, "W3Schools");
```

Una stringa viene convertita in maiuscolo con ***toUpperCase()***: `var text2 = text1.toUpperCase();`

Una stringa viene convertita in minuscolo con ***toLowerCase()***: `var text2 = text1.toLowerCase();`

Il metodo ***trim()*** rimuove gli *spazi bianchi* da entrambi i lati di una stringa: `str.trim()`

Il metodo ***charAt()*** restituisce il carattere in un indice (posizione) specificato in una stringa:

```
var str = "HELLO WORLD";
str.charAt(0); // returns H
```

Numeri - https://www.w3schools.com/js/js_numbers.asp

```
var x = 3.14; // A number with decimals
var y = 3; // A number without decimals

var x = 123e5; // 12300000
var y = 123e-5; // 0.00123
```

JavaScript utilizza l'operatore **+** sia per l'aggiunta che per la concatenazione. I numeri sono sommati. Le stringhe sono concatenate.

```
var x = 10;
var y = 20;
var z = x + y; // z will be 30 (a number)

var x = "10";
var y = "20";
var z = x + y; // z will be 1020 (a string)
```

Se aggiungi un numero e una stringa, il risultato sarà una concatenazione di stringhe:

```
var x = 10;
var y = "20";
var z = x + y; // z will be 1020 (a string)
```

Se aggiungi una stringa e un numero, il risultato sarà una concatenazione di stringhe:

```
var x = "10";
var y = 20;
var z = x + y; // z will be 1020 (a string)
```

L'interprete JavaScript funziona da sinistra a destra. Il primo $10 + 20$ viene aggiunto perché x e y sono entrambi numeri. Quindi $30 + "30"$ è concatenato perché z è una stringa.

JavaScript proverà a convertire stringhe in numeri in tutte le operazioni numeriche. Questo funzionerà:

```
var x = "100";
var y = "10";
var z = x / y; // z will be 10

var x = "100";
var y = "10";
var z = x * y; // z will be 1000
```

```
var x = "100";
var y = "10";
var z = x - y;           // z will be 90
```

Nan - Not a Number

NaN è una parola riservata JavaScript che indica che un numero non è un numero legale.

Cercando di fare aritmetica con una stringa non numerica comporterà NaN(Non un numero):

```
var x = 100 / "Apple"; // x will be NaN (Not a Number)
```

Puoi utilizzare la funzione JavaScript globale **isNaN()** per scoprire se un valore è un numero:

```
var x = 100 / "Apple";
isNaN(x);           // returns true because x is Not a Number
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_numbers_isnan_true

```
typeof NaN;        // returns "number"
```

Infinito

Infinity(o -Infinity) è il valore che JavaScript restituirà se si calcola un numero al di fuori del numero più grande possibile.

```
var myNumber = 2;
while (myNumber != Infinity) { // Execute until Infinity
    myNumber = myNumber * myNumber;
}
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_numbers\_infinity
```

La divisione per 0 (zero) genera anche **Infinity**:

```
var x = 2 / 0;        // x will be Infinity
var y = -2 / 0;       // y will be -Infinity
typeof Infinity;     // returns "number"
```

Normalmente i numeri JavaScript sono valori primitivi creati da letterali:

`var x = 123;`

Ma i numeri possono anche essere definiti come oggetti con la parola chiave **new**:

```
var y = new Number(123);
```

Metodi di numero - https://www.w3schools.com/js/js_number_methods.asp

Il metodo **toString()** restituisce un numero come stringa.

Tutti i metodi numerici possono essere utilizzati su qualsiasi tipo di numeri (letterali, variabili o espressioni):

```
var x = 123;
x.toString();          // returns 123 from variable x
(123).toString();    // returns 123 from literal 123
(100 + 23).toString(); // returns 123 from expression 100 + 23
```

Il metodo **toFixed()** restituisce una stringa, con il numero scritto con un numero specificato di decimali:

```
var x = 9.656;
x.toFixed(0);          // returns 10
x.toFixed(2);          // returns 9.66
x.toFixed(4);          // returns 9.6560
x.toFixed(6);          // returns 9.656000
```

Il metodo **toPrecision()** restituisce una stringa, con un numero scritto con una lunghezza specificata:

```

var x = 9.656;
x.toPrecision();           // returns 9.656
x.toPrecision(2);          // returns 9.7
x.toPrecision(4);          // returns 9.656
x.toPrecision(6);          // returns 9.65600

```

Metodi JavaScript globali

I metodi globali di JavaScript possono essere utilizzati su tutti i tipi di dati JavaScript. Questi sono i metodi più rilevanti quando si lavora con i numeri:

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

Number() può essere usato per convertire le variabili JavaScript in numeri:

```

Number(true);           // returns 1
Number(false);          // returns 0
Number("10");           // returns 10
Number("10.33");         // returns 10.33
Number("10,33");         // returns NaN
Number("10 33");         // returns NaN
Number("John");          // returns NaN

Number(new Date("2017-09-30")); // returns 1506729600000

```

parseInt() analizza una stringa e restituisce un numero intero. Gli spazi sono ammessi. Viene restituito solo il primo numero:

```

parseInt("10");          // returns 10
parseInt("10.33");        // returns 10
parseInt("10 20 30");     // returns 10
parseInt("10 years");      // returns 10
parseInt("years 10");       // returns NaN

```

parseFloat() analizza una stringa e restituisce un numero. Gli spazi sono ammessi. Viene restituito solo il primo numero:

```

parseFloat("10");          // returns 10
parseFloat("10.33");        // returns 10.33
parseFloat("10 20 30");     // returns 10
parseFloat("10 years");      // returns 10
parseFloat("years 10");       // returns NaN

```

Array - https://www.w3schools.com/js/js_arrays.asp

Gli array JavaScript sono scritti con parentesi quadre, separati da virgole.

```
var cars=["Saab","Volvo","BMW"];
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_array

L'utilizzo di un array letterale è il modo più semplice per creare una matrice JavaScript.

Sintassi: `var array_name = [item1, item2, ...];`

Si accede a un elemento dell'array facendo riferimento al **numero di indice**.

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];

https://www.w3schools.com/js/tryit.asp?filename=tryjs\_array\_element\_change
```

Con JavaScript, è possibile accedere all'intero array facendo riferimento al nome dell'array:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;

https://www.w3schools.com/js/tryit.asp?filename=tryjs\_array\_full
```

Gli array sono un tipo speciale di oggetti. L'operatore **typeof** in JavaScript restituisce "oggetto" per gli array.

Puoi avere oggetti in un array. Puoi avere funzioni in un array. Puoi avere più array in un array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Proprietà e metodi dell'array

```
var x = cars.length; // The length property returns the number of elements
var y = cars.sort(); // The sort() method sorts arrays
```

La proprietà **length** di un array restituisce la lunghezza di un array (il numero di elementi dell'array).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length; // the length of fruits is 4
```

Il modo più semplice per aggiungere un nuovo elemento a un array è utilizzare il metodo **push()**:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon"); // adds a new element (Lemon) to fruits

typeof fruits; // returns object

Array.isArray(fruits); // returns true
```

Il metodo **pop()** rimuove l'ultimo elemento da un array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop(); // Removes the last element ("Mango") from fruits
```

Eventi - https://www.w3schools.com/js/js_events.asp

Gli eventi HTML sono "cose" che accadono agli **elementi** HTML.

Un evento HTML può essere qualcosa che fa il browser o qualcosa che un utente fa. Es:

- Una pagina Web HTML ha finito il caricamento
- È stato modificato un campo di input HTML
- È stato fatto clic su un pulsante HTML

JavaScript ti consente di eseguire il codice quando vengono rilevati eventi.

HTML consente agli attributi del gestore eventi, con **codice JavaScript**, di essere aggiunti agli elementi HTML.

```
<element event="some JavaScript"> o <element event='some JavaScript'>
```

Nell'esempio seguente, un attributo **onclick** (con codice), viene aggiunto a un elemento **<button>**:

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick1

Il codice cambia il contenuto del proprio elemento (usando **this.innerHTML**):

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick

Il codice JavaScript è spesso lungo diverse righe. È più comune vedere le funzioni di chiamata degli attributi degli eventi:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_events1\

Evento	Descrizione
onchange	Un elemento HTML è stato modificato
onclick	L'utente fa clic su un elemento HTML
onmouseover	L'utente sposta il mouse su un elemento HTML
onmouseout	L'utente sposta il mouse lontano da un elemento HTML
onkeydown	L'utente preme un tasto della tastiera
onload	Il browser ha finito di caricare la pagina

Onkeyup:

L'evento onkeyup si verifica quando l'utente rilascia un tasto (sulla tastiera):

```
object.onkeyup = function(){myScript};
```

```
<input type="text" onkeyup="myFunction()">
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onkeyup

Onload:

Esegui un JavaScript subito dopo aver caricato una pagina:

```
object.onload = function(){myScript};
```

```
<body onload="myFunction()">
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onload

L'**onload** è spesso usato all'interno dell'elemento <body> per eseguire uno script una volta che una pagina web ha completamente caricato tutto il contenuto (incluse immagini, file di script, file CSS, ecc.).

L'evento **onload** può essere utilizzato anche per gestire i *cookie*.

Onmouseover/Onmouseout:

L'evento **onmouseover** si verifica quando il puntatore del mouse viene spostato su un elemento o su uno dei suoi figli:

```
object.onmouseover = function(){myScript};
```

L'evento **onmouseout** si verifica quando il puntatore del mouse viene spostato da un elemento o fuori da uno dei suoi figli:

```
object.onmouseout = function(){myScript};
```

```

```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onmouseover

onblur:

Esegui un JavaScript quando un utente lascia un campo di input:

```
object.onblur = function(){myScript};
```

```
<input type="text" onblur="myFunction()">
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onblur

L'evento **onblur** si verifica quando un oggetto perde lo stato attivo, spesso utilizzato con il codice di convalida del modulo. l'evento **onblur** è l'opposto dell'evento **onfocus** .

onfocus:

Esegui un JavaScript quando viene attivato un campo di input:

```
object.onfocus = function(){myScript};  
<input type="text" onfocus="myFunction()">  
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_onfocus
```

L'evento **onfocus** si verifica quando un elemento diventa attivo, viene spesso utilizzato con <input>, <select> e <a>.

onclick:

Esegui un JavaScript quando si fa clic su un pulsante:

```
object.onclick = function(){myScript};  
<button onclick="myFunction()">Click me</button>  
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_onclick
```

onreset:

Esegui un JavaScript quando viene resettato un modulo: `object.onreset = function(){myScript};`

```
<form onreset="myFunction()">  
  Enter name: <input type="text">  
  <input type="reset">  
</form> https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_onreset
```

onsubmit:

Esegui un JavaScript quando viene inviato un modulo:

```
<form onsubmit="myFunction()">  
  Enter name: <input type="text">  
  <input type="submit">  
</form> https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_onsubmit
```

Metodi

setInterval:

Chiama una funzione o valuta un'espressione a intervalli specificati (in millisecondi).

Il metodo **setInterval ()** continuerà a chiamare la funzione finché non viene chiamato **clearInterval ()** o la finestra viene chiusa.

Il valore ID restituito da **setInterval ()** viene utilizzato come parametro per il metodo **clearInterval ()**.

Suggerimento: 1000 ms = 1 secondo.

Suggerimento: per eseguire una funzione solo una volta, dopo un numero specificato di millisecondi, utilizzare il metodo **setTimeout ()** .

```
setInterval(function(){ alert("Hello"); }, 3000);  
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_win\_setinterval
```

clearInterval:

Il metodo **clearInterval ()** cancella un timer impostato con il metodo **setInterval ()** .

Il valore ID restituito da ***setInterval*** () viene utilizzato come parametro per il metodo ***clearInterval*** ().

Nota: per poter utilizzare il metodo ***clearInterval*** (), è necessario utilizzare una variabile durante la creazione del metodo dell'intervallo:

```
myVar = setInterval("javascript function", milliseconds);
```

Quindi sarai in grado di interrompere l'esecuzione chiamando il metodo ***clearInterval*** ().

```
clearInterval(myVar);
```

blur:

Il metodo ***blur*** () viene utilizzato per rimuovere lo stato attivo da un elemento.

Suggerimento: usa il metodo ***focus*** () per dare fuoco a un elemento.

Rimuovi lo stato attivo da un elemento <a>: document.getElementById("myAnchor").blur();

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_html_blur

focus:

Il metodo ***focus*** () è usato per focalizzare un elemento (se può essere focalizzato).

Suggerimento: usa il metodo ***blur*** () per rimuovere lo stato attivo da un elemento.

Dà attenzione a un elemento <a>: document.getElementById("myAnchor").focus();

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_html_blur

click:

Il metodo ***click*** () simula un clic del mouse su un elemento.

Questo metodo può essere utilizzato per eseguire un clic su un elemento come se l'utente avesse fatto clic manualmente su di esso.

```
document.getElementById("myCheck").click(); // Click on the checkbox
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_html_click

reset:

Reimposta i valori di tutti gli elementi in un modulo (come se si stesse facendo clic sul pulsante Ripristina):

```
document.getElementById("myForm").reset();
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_form_reset

submit:

Il metodo ***submit*** () invia il modulo (come per il pulsante Invia).

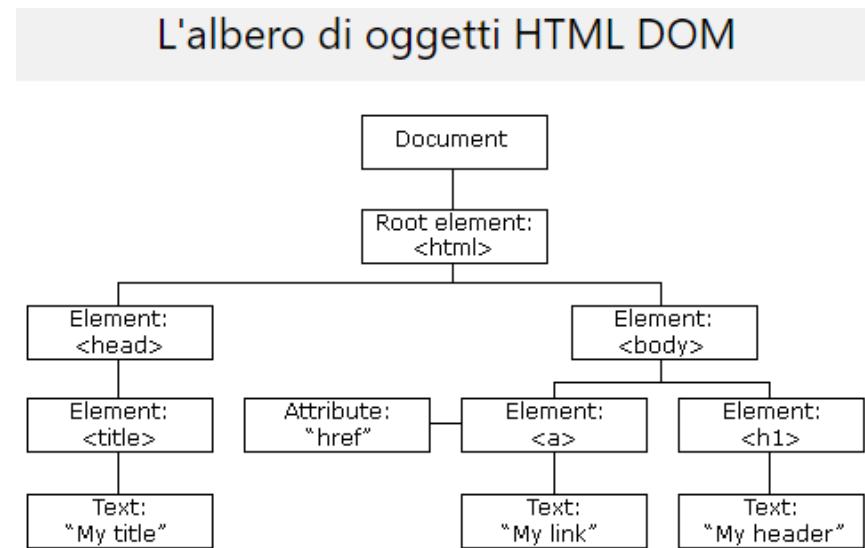
```
document.getElementById("myForm").submit();
```

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_form_submit

Con il DOM HTML, JavaScript può accedere e modificare tutti gli elementi di un documento HTML.

Quando una pagina Web viene caricata, il browser crea un **Document Object Model** della pagina.

Il modello **HTML DOM** è costruito come un albero di **oggetti** :



- JavaScript può cambiare tutti gli elementi HTML nella pagina
- JavaScript può cambiare tutti gli attributi HTML nella pagina
- JavaScript può cambiare tutti gli stili CSS nella pagina
- JavaScript può rimuovere elementi e attributi HTML esistenti
- JavaScript può aggiungere nuovi elementi e attributi HTML
- JavaScript può reagire a tutti gli eventi HTML esistenti nella pagina
- JavaScript può creare nuovi eventi HTML nella pagina

HTML DOM è un modello di **oggetto** standard e **un'interfaccia di programmazione** per HTML. Definisce:

- Gli elementi HTML come **oggetti**
- Le **proprietà** di tutti gli elementi HTML
- I **metodi** per accedere a tutti gli elementi HTML
- Gli **eventi** per tutti gli elementi HTML

HTML DOM è uno standard su come ottenere, modificare, aggiungere o eliminare elementi HTML.

Metodi DOM HTML - https://www.w3schools.com/js/js_htmldom_methods.asp

I metodi DOM HTML sono **azioni** che puoi eseguire su elementi HTML. Le proprietà DOM HTML sono **valori** che è possibile impostare o modificare. È possibile accedere al **DOM HTML** con **JavaScript** (e con altri linguaggi).

Nel DOM, tutti gli elementi HTML sono definiti come **oggetti**. **L'interfaccia di programmazione** è proprietà e metodi di ciascun oggetto.

Una **proprietà** è un **valore** che puoi ottenere o impostare (ad esempio modificare il contenuto di un elemento).

Un **metodo** è **un'azione** che puoi fare (come aggiungere o eliminare un elemento HTML).

L'esempio seguente modifica il contenuto (**innerHTML**) **<p>**dell'elemento con **id="demo"**:

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_dom\_method
```

Il metodo getElementById:

Il modo più comune per accedere a un elemento HTML è utilizzare l'elemento ***id***.

Nell'esempio sopra il metodo ***getElementById*** utilizzato ***id="demo"*** per trovare l'elemento.

La proprietà innerHTML

Il modo più semplice per ottenere il contenuto di un elemento è usando la proprietà ***innerHTML***, è utile per ottenere o sostituire il contenuto di elementi HTML, può essere utilizzata per ottenere o modificare qualsiasi elemento HTML, inclusi <html> e <body>.

HTML DOM Document - https://www.w3schools.com/js/js_htmldom_document.asp

L'oggetto ***document*** è il proprietario di tutti gli altri oggetti nella tua pagina web.

Se si desidera accedere a qualsiasi elemento in una pagina HTML, si inizia con l'accesso all'oggetto ***document***.

Trovare elementi HTML:

<code>document.getElementById(<i>id</i>)</code>	Find an element by element id
<code>document.getElementsByTagName(<i>name</i>)</code>	Find elements by tag name
<code>document.getElementsByClassName(<i>name</i>)</code>	Find elements by class name

Modifica degli elementi HTML:

Property	Description
<code>element.innerHTML = new html content</code>	Change the inner HTML of an element
<code>element.setAttribute(<i>attribute</i>, <i>value</i>)</code>	Change the attribute value of an HTML element
<code>element.style.property = new style</code>	Change the style of an HTML element
Method	Description
<code>element.setAttribute(<i>attribute</i>, <i>value</i>)</code>	Change the attribute value of an HTML element

Aggiunta ed eliminazione di elementi:

<code>document.createElement(<i>element</i>)</code>	Create an HTML element
<code>document.removeChild(<i>element</i>)</code>	Remove an HTML element
<code>document.appendChild(<i>element</i>)</code>	Add an HTML element
<code>document.replaceChild(<i>new</i>, <i>old</i>)</code>	Replace an HTML element
<code>document.write(<i>text</i>)</code>	Write into the HTML output stream

Aggiunta di gestori di eventi:

Aggiunta del codice del gestore eventi a un evento onclick:

```
document.getElementById(id).onclick = function(){code}
```

Elementi DOM HTML - https://www.w3schools.com/js/js_htmldom_elements.asp

Spesso, con JavaScript, vuoi manipolare elementi HTML. Per fare ciò, devi prima trovare gli elementi. Ci sono diversi modi per farlo:

- Trovare elementi HTML per ***id***

```
var myElement = document.getElementById("intro");  
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_dom\_getelementbyid
```

- Ricerca di elementi HTML in base al ***nome del tag***

```
var x = document.getElementById("main");  
var y = x.getElementsByTagName("p");  
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_dom\_getelementsbytagname2
```

- Ricerca di elementi HTML in base al ***nome della classe***

```
var x = document.getElementsByClassName("intro");  
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_dom\_getelementsbyclassname
```

- Trovare elementi HTML dai ***selettori CSS***

```
var x = document.querySelectorAll("p.intro");  
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_dom\_queryselectorall
```

Modifica HTML - https://www.w3schools.com/js/js_htmldom_html.asp

JavaScript può creare contenuti HTML dinamici. In JavaScript, ***document.write()*** può essere utilizzato per scrivere direttamente nel ***flusso di output*** HTML:

```
<script>  
document.write(Date());  
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_date
```

Il modo più semplice per modificare il ***contenuto di un elemento*** HTML consiste nell'utilizzare la proprietà ***innerHTML***.

```
<p id="p1">Hello World!</p>  
<script>  
document.getElementById("p1").innerHTML = "New text!";  
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_change\_innerhtml
```

Per cambiare il ***valore di un attributo*** HTML, usa questa sintassi:

```
document.getElementById(id).attribute = new value  
  
  
<script>  
document.getElementById("myImage").src = "landscape.jpg";  
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_dom\_image
```

Modifica CSS - https://www.w3schools.com/js/js_htmldom_css.asp

Il DOM HTML consente a JavaScript di modificare lo ***stile degli elementi*** HTML.

```
document.getElementById(id).style.property = new style  
  
<p id="p2">Hello World!</p>  
<script>  
document.getElementById("p2").style.color = "blue";
```

<p>The paragraph above was changed by a script.</p>

HTML DOM ti consente di eseguire codice quando si verifica un **evento**.

Gli eventi sono generati dal browser quando "le cose accadono" agli elementi HTML:

- Si fa clic su un elemento
- La pagina è stata caricata
- I campi di input sono cambiati

<h1 id="id1">My Heading 1</h1>

<button type="button"

onclick="document.getElementById('id1').style.color = 'red'">

Click Me!</button>

https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_color2

Eventi HTML DOM - https://www.w3schools.com/js/js_htmldom_events.asp

In questo esempio, il contenuto dell'elemento <h1> viene modificato quando un utente fa clic su di esso:

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

https://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick2

In questo esempio, una funzione viene chiamata dal gestore eventi:

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>

function changeText(id) {
 id.innerHTML = "Ooops!";

}

</script>

https://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick3

Attributi evento HTML

Per assegnare eventi a elementi HTML è possibile utilizzare gli attributi evento.

<button onclick="displayDate()">Try it</button>

https://www.w3schools.com/js/tryit.asp?filename=tryjs_events1

Assegna eventi utilizzando il DOM HTML

Il DOM HTML ti consente di assegnare eventi ad elementi HTML usando JavaScript:

<script>

document.getElementById("myBtn").onclick = displayDate;

</script>

https://www.w3schools.com/js/tryit.asp?filename=tryjs_events2

Gli eventi

Gli eventi **onload e onunload** vengono attivati quando l'utente entra o esce dalla pagina.

L'evento **onload** può essere utilizzato per verificare il tipo di browser del visitatore e la versione del browser e caricare la versione corretta della pagina Web in base alle informazioni.

Gli eventi **onload e onunload** possono essere utilizzati per gestire i **cookie**.

<body onload="checkCookies()">

https://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onload

L'evento **onchange** è spesso utilizzato in combinazione con la convalida dei campi di input.

```
<input type="text" id="fname" onchange="UpperCase()">
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_onchange

Gli eventi **onmouseover** e **onmouseout** possono essere utilizzati per attivare una funzione quando l'utente esegue il mouse sopra o fuori da un elemento HTML:

https://www.w3schools.com/js/tryit.asp?filename=tryjs_events_mouseover

I **onmousedown**, **onmouseup** e **onclick** gli eventi sono tutti parti di un clic del mouse. Per prima cosa quando viene cliccato un pulsante del mouse, viene attivato l'evento **onmousedown**, quindi, quando viene rilasciato il pulsante del mouse, viene attivato l'evento **onmouseup**, infine, quando viene completato il clic del mouse, viene attivato l'evento **onclick**.

HTML EventListener - https://www.w3schools.com/js/js_htmldom_eventlistener.asp

VALIDAZIONE DEL FORM - https://www.w3schools.com/js/js_validation.asp

Uno degli utilizzi più frequenti di JavaScript è nell'ambito della validazione dei campi di un form:

- Riduce il carico delle applicazioni server side filtrando l'input
- Riduce il ritardo in caso di errori di inserimento dell'utente
- Consente di introdurre dinamicità all'interfaccia web

Generalmente si valida un form in due momenti:

1. Durante l'inserimento utilizzando l'evento **onChange()** sui vari controlli
2. Al momento del **submit** di un **form**. Intercettare l'evento **onClick()** del bottone di **submit** o l'evento **onSubmit()** del form, assegnare all'attributo il valore «*return validationFunc()*»

La funzione di validazione deve:

- Controllare se il valore di un dato campo corrisponde a quanto ci si aspetta, restituendo true e mandando il form alla servlet.
- In caso negativo, dare un messaggio di errore, restituendo false e non mandando il modulo.

Se un campo modulo (**fname**) è vuoto, questa funzione avvisa un messaggio e restituisce false, per impedire che il modulo venga inviato:

```
function validateForm() {  
    var x = document.forms["myForm"]["fname"].value;  
    if (x == "") {  
        alert("Name must be filled out");  
        return false;  
    }  
}
```

La funzione può essere chiamata quando il modulo viene inviato:

```
<form name="myForm" action="/action_page.php" onSubmit="return validateForm()" method="post">  
Name: <input type="text" name="fname">  
<input type="submit" value="Submit">  
</form> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_validation\_js
```

Convalida automatica del modulo HTML

La convalida del modulo HTML può essere eseguita automaticamente dal browser.

Se un campo modulo (**fname**) è vuoto, l'attributo **required** impedisce che questo modulo venga inviato:

```
<form action="/action_page.php" method="post">  
    <input type="text" name="fname" required>  
    <input type="submit" value="Submit">  
</form> https://www.w3schools.com/js/tryit.asp?filename=tryjs\_validation\_html
```

Molto spesso, lo scopo della convalida dei dati è quello di garantire un corretto inserimento dell'utente.

La convalida del lato server viene eseguita da un server Web, dopo che l'input è stato inviato al server.

La convalida lato client viene eseguita da un browser Web, prima che l'input venga inviato a un server Web.

Convalida vincoli HTML

La convalida del vincolo HTML si basa su:

- **Attributi di input HTML** di convalida del vincolo
- **Selezione vincoli CSS Pseudo selettori**
- **Proprietà e metodi DOM** di convalida del vincolo

Attributi di input HTML di convalida vincoli:

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

https://www.w3schools.com/html/html_form_attributes.asp

Selettori Pseudo CSS di convalida vincoli:

Selector	Description
:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

https://www.w3schools.com/css/css_pseudo_classes.asp

API di convalida JavaScript - https://www.w3schools.com/js/js_validation_api.asp

Metodi DOM Validazione vincoli:

Property	Description
checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validationMessage property of an input element.

Se un campo di input contiene dati non validi, visualizza un messaggio:

```

<input id="id1" type="number" min="100" max="300" required>
<button onclick="myFunction()">OK</button>
<p id="demo"></p>
<script>  function myFunction() {
    var inpObj = document.getElementById("id1");
    if (!inpObj.checkValidity()) {
        document.getElementById("demo").innerHTML = inpObj.validationMessage;
    }
}</script>  
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_check

JavaScript Espressioni regolari - https://www.w3schools.com/js/js_regexp.asp

Un'espressione regolare è una sequenza di caratteri che forma un **modello di ricerca**, che può essere utilizzato per la ricerca del testo e le operazioni di sostituzione del testo.

Le espressioni regolari possono essere utilizzate per eseguire tutti i tipi di **ricerca di testo** e operazioni di **sostituzione del testo**.

Sintassi: */pattern/modifiers;*

`var patt = /w3schools/i;`

w3schools i è un'espressione regolare.

w3schools è un pattern (da utilizzare in una ricerca).

i è un modificatore (modifica la ricerca senza distinzione tra maiuscole e minuscole).

In JavaScript, le espressioni regolari vengono utilizzate con i due **metodi stringa**:

- Il metodo **search()** utilizza un'espressione per cercare una corrispondenza e restituisce la posizione della corrispondenza con una stringa:

```
var str = "Visit W3Schools!";
var n = str.search("W3Schools");
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_string\_search
```

con una espressione regolare:

```
var str = "Visit W3Schools";
var n = str.search(/w3schools/i);
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_regexp\_string\_search
```

- Il metodo **replace()** restituisce una stringa modificata in cui il modello viene sostituito.

con una stringa:

```
var str = "Visit Microsoft!";
var res = str.replace("Microsoft", "W3Schools");
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_string\_replace
```

con una espressione regolare:

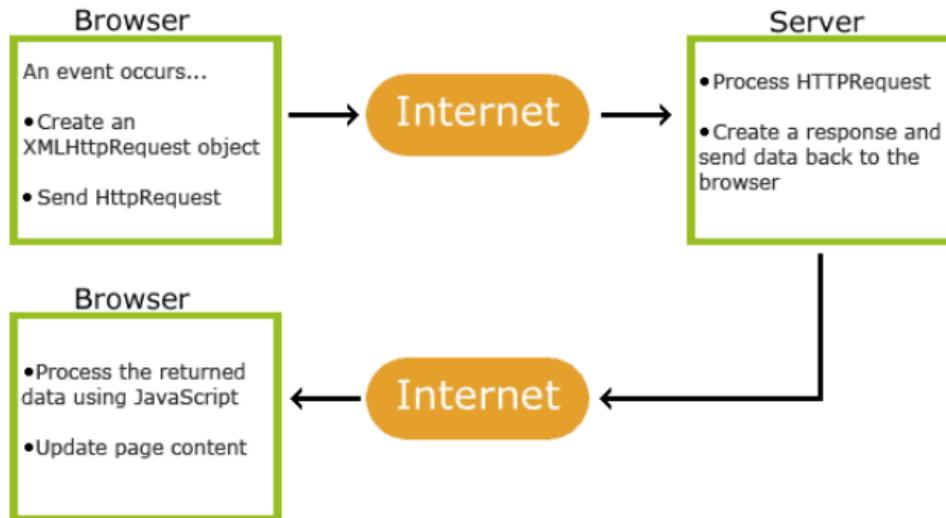
```
var str = "Visit Microsoft!";
var res = str.replace(/microsoft/i, "W3Schools");
https://www.w3schools.com/js/tryit.asp?filename=tryjs\_regexp\_string\_replace
```

AJAX (Asynchronous JavaScript And XML) - https://www.w3schools.com/js/js_ajax_intro.asp

- Leggi i dati da un server web - dopo che la pagina è stata caricata
- Aggiorna una pagina Web senza ricaricare la pagina
- Invia dati a un server web - in background

AJAX utilizza solo una combinazione di:

- Un oggetto **XMLHttpRequest** integrato nel browser (per richiedere dati da un server Web)
- **DOM JavaScript e HTML** (per visualizzare o utilizzare i dati)



1. Un evento si verifica in una pagina Web (la pagina viene caricata, si fa clic su un pulsante)
2. Un oggetto XMLHttpRequest viene creato da JavaScript
3. L'oggetto XMLHttpRequest invia una richiesta a un server web
4. Il server elabora la richiesta
5. Il server invia una risposta alla pagina web
6. La risposta viene letta da JavaScript
7. L'azione corretta (come l'aggiornamento della pagina) viene eseguita da JavaScript

Oggetto XMLHttpRequest - https://www.w3schools.com/js/js_ajax_http.asp

L'oggetto **XMLHttpRequest** può essere utilizzato per scambiare dati con un server Web dietro le quinte. Ciò significa che è possibile aggiornare parti di una pagina Web senza ricaricare l'intera pagina.

Sintassi per la creazione di un oggetto **XMLHttpRequest**: `var xhttp = new XMLHttpRequest();`

Per motivi di sicurezza, i browser moderni non consentono l'accesso ai domini.

Ciò significa che sia la pagina Web che il file XML che tenta di caricare devono trovarsi sullo stesso server.

Metodi oggetto XMLHttpRequest:

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(<i>method, url, async, user, psw</i>)	Specifies the request <i>method</i> : the request type GET or POST <i>url</i> : the file location <i>async</i> : true (asynchronous) or false (synchronous) <i>user</i> : optional user name <i>psw</i> : optional password
send()	Sends the request to the server Used for GET requests
send(<i>string</i>)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

Proprietà dell'oggetto XMLHttpRequest:

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

AJAX Request - https://www.w3schools.com/js/js_ajax_http_send.asp

Per inviare una richiesta a un server, utilizziamo i metodi **open()** e **send()** XMLHttpRequest dell'oggetto:

Col GET:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Potresti ottenere un risultato memorizzato nella cache. Per evitare ciò, aggiungi un ID univoco all'URL:

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random(), true);
xhttp.send();
```

Se si desidera inviare informazioni con il metodo **GET**, aggiungere le informazioni all'URL:

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);  
xhttp.send();
```

Col POST:

```
xhttp.open("POST", "demo_post.asp", true);  
xhttp.send();
```

Per i dati POST come un modulo HTML, aggiungere un'intestazione HTTP con **setRequestHeader()**. Specificare i dati che si desidera inviare nel metodo **send()**:

```
xhttp.open("POST", "ajax_test.asp", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xhttp.send("fname=Henry&lname=Ford");
```

L'URL:

Il parametro **url** del metodo **open()** è un indirizzo di un file su un server. Il file può essere qualsiasi tipo di file, come .txt e .xml, o file di scripting del server come .asp e .php (che possono eseguire azioni sul server prima di inviare la risposta).

Asincrono:

Le richieste del server devono essere inviate in modo asincrono. Il parametro **async** del metodo **open()** deve essere impostato su **true**:

```
xhttp.open("GET", "ajax_test.asp", true);
```

Inviando in modo asincrono, il JavaScript non deve attendere la risposta del server, ma può invece:

- eseguire altri script mentre si attende la risposta del server
- affrontare la risposta dopo che la risposta è pronta

AJAX Response - https://www.w3schools.com/js/js_ajax_http_response.asp

La proprietà **readyState** contiene lo stato di **XMLHttpRequest**.

*La proprietà **onreadystatechange** definisce una funzione da eseguire quando il **readyState** cambia, eseguendo una eventuale funzione di callback.*

La proprietà **status** e la proprietà **statusText** mantengono lo stato dell'oggetto **XMLHttpRequest**.

onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Quando **readyState** è 4 e lo stato è 200, la risposta è pronta:

L' evento **onreadystatechange** viene attivato quattro volte (1-4), una volta per ogni modifica nel **readyState**.

Utilizzo di una funzione di richiamata:

Una funzione di **callback** è una funzione passata come parametro ad un'altra funzione.

Se si dispone di più di un'attività AJAX in un sito Web, è necessario creare una funzione per l'esecuzione **XMLHttpRequest** dell'oggetto e una funzione di **callback** per ciascuna attività AJAX.

La chiamata di funzione dovrebbe contenere l'URL e quale funzione chiamare quando la risposta è pronta.

```
loadDoc("url-1", myFunction1);

function loadDoc(url, cFunction) {
  var xhttp;
  xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      cFunction(this);
    }
  };
  xhttp.open("GET", url, true);
  xhttp.send();
}
function myFunction1(xhttp) {
  // action goes here
}
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_ajax_callback

La proprietà **responseText** restituisce la risposta del server come una stringa JavaScript e può essere utilizzata di conseguenza: `document.getElementById("demo").innerHTML = xhttp.responseText;`

Il metodo **getAllResponseHeaders()** restituisce tutte le informazioni di intestazione dalla risposta del server.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML =
      this.getAllResponseHeaders();
  }
}
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_ajax_header

Il metodo **getResponseHeader()** restituisce informazioni specifiche sull'intestazione dalla risposta del server.

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML =
      this.getResponseHeader("Last-Modified");
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

https://www.w3schools.com/js/tryit.asp?filename=tryjs_ajax_lastmodified

Domanda 22 (points: from 0 to 4) Completare il codice della funzione loadDoc(), facendo in modo che alla pressione del bottone "Change Content" il contenuto del div venga sostituito con il contenuto del file all'URL "/ajax.txt".

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {

    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "/ajax.txt", true);
    xhttp.send();
}

</script>
</body>
</html>
```

JSON: JavaScript Object Notation- https://www.w3schools.com/js/js_json_intro.asp

- JSON è una sintassi per l'archiviazione e lo scambio di dati.
- JSON è un testo, scritto con notazione di oggetti JavaScript.

Scambio di dati:

Quando si scambiano dati tra un browser e un server, i dati possono essere solo testo.

JSON è testo e possiamo convertire qualsiasi oggetto JavaScript in JSON e inviare JSON al server. Possiamo inoltre convertire qualsiasi JSON ricevuto dal server in oggetti JavaScript.

Invio di dati:

Se si dispone di dati memorizzati in un oggetto JavaScript, è possibile convertire l'oggetto in JSON e inviarlo a un server:

```
var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_send

Ricevendo dati:

Se si ricevono dati in formato JSON, è possibile convertirli in un oggetto JavaScript:

```
var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_receive

Memorizzazione dei dati:

Quando si memorizzano i dati, i dati devono essere di un certo formato e, indipendentemente da dove si sceglie di memorizzarli, il **testo** è sempre uno dei formati legali. **JSON** rende possibile memorizzare **oggetti JavaScript** come testo. Memorizzazione dei dati nella memoria locale:

```
// Storing data:
myObj = {name: "John", age: 31, city: "New York"};
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
```

```
// Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_store

Sintassi - https://www.w3schools.com/js/js_json_syntax.asp

La sintassi JSON è derivata dalla sintassi di notazione di oggetti JavaScript:

- I dati sono in coppie nome / valore: "name": "John"
- I dati sono separati da virgole
- Le parentesi graffe tengono oggetti: { "name": "John" }
- Le parentesi quadre contengono array

I nomi JSON richiedono doppi apici. I nomi JavaScript non lo fanno.

I **valori** devono essere uno dei seguenti tipi di dati:

- una stringa { "name": "John" }
- un numero { "age": 30 }
- un oggetto (oggetto JSON) { "employee": { "name": "John", "age": 30 } }
- un array { "employees": ["John", "Anna", "Peter"] }
- un booleano { "sale": true }
- nullo { "middleName": null }

JSON vs XML - https://www.w3schools.com/js/js_json_xml.asp

Sia JSON che XML possono essere utilizzati per ricevere dati da un server web.

JSON:

```
{"employees": [  
    { "firstName": "John", "lastName": "Doe" },  
    { "firstName": "Anna", "lastName": "Smith" },  
    { "firstName": "Peter", "lastName": "Jones" }  
]
```

XML:

```
<employees>  
    <employee>  
        <firstName>John</firstName> <lastName>Doe</lastName>  
    </employee>  
    <employee>  
        <firstName>Anna</firstName> <lastName>Smith</lastName>  
    </employee>  
    <employee>  
        <firstName>Peter</firstName> <lastName>Jones</lastName>  
    </employee>  
</employees>
```

- Sia JSON che XML sono "auto descrittivi" (leggibili dall'uomo)
- Sia JSON che XML sono gerarchici (valori all'interno dei valori)
- Sia JSON che XML possono essere analizzati e utilizzati da molti linguaggi di programmazione
- Sia JSON che XML possono essere recuperati con XMLHttpRequest

Unica differenza :

- JSON non usa il tag di fine
- JSON è più breve
- JSON è più veloce da leggere e scrivere
- JSON può usare matrici
- **XML** deve essere analizzato con un **parser XML**.
- **JSON** può essere analizzato da una **funzione JavaScript** standard.

XML è molto più difficile da analizzare rispetto a JSON e viene analizzato in un oggetto JavaScript pronto per l'uso.

Oggetti JSON - https://www.w3schools.com/js/js_json_objects.asp

Gli oggetti JSON sono circondati da parentesi graffe {}, coppie nome / valore, i nomi devono essere stringhe e i valori devono essere un tipo di dati JSON valido: { "name": "John", "age": 30, "car": null }

Accesso ai valori degli oggetti:

```
myObj = { "name": "John", "age": 30, "car": null };
x = myObj.name;                                     https://www.w3schools.com/js/tryit.asp?filename=tryjson\_object\_dot
or x = myObj["name"];
```

Loop di un oggetto:

È possibile scorrere le **proprietà** dell'oggetto utilizzando il ciclo for-in:

```
myObj = { "name": "John", "age": 30, "car": null };
for (x in myObj) {
    document.getElementById("demo").innerHTML += x;
}                                                 https://www.w3schools.com/js/tryit.asp?filename=tryjson\_object\_loop
```

In un ciclo for-in, utilizzare la notazione della parentesi **per accedere ai valori** delle proprietà:

```
myObj = { "name": "John", "age": 30, "car": null };
for (x in myObj) {
    document.getElementById("demo").innerHTML += myObj[x];
}                                                 https://www.w3schools.com/js/tryit.asp?filename=tryjson\_object\_loop\_bracket
```

Oggetti JSON annidati:

I valori in un oggetto JSON possono essere un altro oggetto JSON:

```
myObj = {
    "name": "John",
    "cars": {
        "car1": "Ford",
        "car2": "BMW"
    }
}
```

È possibile accedere agli oggetti JSON nidificati usando la notazione dot o la notazione parentesi:

```
x = myObj.cars.car2;
// or:
x = myObj.cars["car2"];      https://www.w3schools.com/js/tryit.asp?filename=tryjson\_object\_nested
```

Modifica valori:

Puoi usare la notazione a punti per modificare qualsiasi valore in un oggetto JSON:

```
myObj.cars.car2 = "Mercedes"; https://www.w3schools.com/js/tryit.asp?filename=tryjson\_object\_modify
```

Elimina proprietà oggetto:

Usa la parola **delete** chiave per cancellare le proprietà da un oggetto JSON:

```
delete myObj.cars.car2;      https://www.w3schools.com/js/tryit.asp?filename=tryjson\_object\_delete
```

JSON.parse() - https://www.w3schools.com/js/js_json_parse.asp

Quando si ricevono dati da un server Web, i dati sono sempre una stringa. Analizza i dati con **JSON.parse()** e i dati diventano oggetti JavaScript.

Immagina di aver ricevuto questo testo da un server web:

```
'{"name": "John", "age": 30, "city": "New York"}'
```

*Usa la funzione JavaScript **JSON.parse()** per convertire il testo in un oggetto JavaScript:*

```
var obj = JSON.parse('{"name": "John", "age": 30, "city": "New York"}');
```

Usa l'oggetto JavaScript nella tua pagina:

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
</script> https://www.w3schools.com/js/tryit.asp?filename=tryjson\_parse
```

JSON dal server:

È possibile richiedere JSON dal server utilizzando una richiesta AJAX. Finché la risposta dal server è scritta in formato JSON, è possibile analizzare la stringa in un oggetto JavaScript.

Utilizzare **XMLHttpRequest** per ottenere dati dal server:

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    var myObj = JSON.parse(this.responseText);
    document.getElementById("demo").innerHTML = myObj.name;
  }
};
xmlhttp.open("GET", "json_demo.txt", true);
xmlhttp.send(); https://www.w3schools.com/js/tryit.asp?filename=tryjson\_ajax
```

Eccezioni:

Gli oggetti data non sono consentiti in JSON. Se è necessario includere una data, scriverla come una stringa.

Convertire una stringa in una data:

```
var text = '{ "name": "John", "birth": "1986-12-14", "city": "New York"}';
var obj = JSON.parse(text);
obj.birth = new Date(obj.birth);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;

https://www.w3schools.com/js/tryit.asp?filename=tryjson\_parse\_date
```

Funzioni:

Le funzioni non sono consentite in JSON. Se è necessario includere una funzione, scriverla come una stringa.

Converti una stringa in una funzione:

```
var text = '{ "name": "John", "age": "function () {return 30;}", "city": "New York"}';
var obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();

https://www.w3schools.com/js/tryit.asp?filename=tryjson\_parse\_function
```

JSON.stringify() - https://www.w3schools.com/js/js_json_stringify.asp

Quando si inviano dati a un server Web, i dati devono essere una stringa. Convertire un oggetto JavaScript in una stringa con **JSON.stringify()**.

Immagina di avere questo oggetto in JavaScript:

```
var obj = { name: "John", age: 30, city: "New York" };
```

Usa la funzione JavaScript **JSON.stringify()** per convertirla in una stringa.

```
var myJSON = JSON.stringify(obj);
```

myJSON è ora una stringa e pronta per essere inviata a un server:

```
document.getElementById("demo").innerHTML = myJSON;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_stringify

Eccezioni:

In JSON, gli oggetti data non sono consentiti. La funzione **JSON.stringify()** convertirà qualsiasi data in stringhe.

```
var obj = { name: "John", today: new Date(), city : "New York" };
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_stringify_date

Funzioni:

La funzione **JSON.stringify()** rimuoverà qualsiasi funzione da un oggetto JavaScript, sia la chiave che il valore:

```
var obj = { name: "John", age: function () {return 30;}, city: "New York"};
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_stringify_function

Questo può essere omesso se converti le tue funzioni in stringhe prima di eseguire la funzione **JSON.stringify()**.

```
var obj = { name: "John", age: function () {return 30;}, city: "New York"};
obj.age = obj.age.toString();
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

HTML JSON - https://www.w3schools.com/js/js_json_html.asp

JSON può essere facilmente tradotto in JavaScript, usato per creare HTML nelle tue pagine web.

Tabella HTML dinamica:

Crea la tabella HTML in base al valore di un menu a discesa:

```
<select id="myselect" onchange="change_myselect(this.value)">
  <option value="">Choose an option:</option>
  <option value="customers">Customers</option>
  <option value="products">Products</option>
  <option value="suppliers">Suppliers</option>
</select>
<script>
function change_myselect(sel) {
  var obj, dbParam, xmlhttp, myObj, x, txt = "";
  obj = { table: sel, limit: 20 };
  dbParam = JSON.stringify(obj);
  xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      myObj = JSON.parse(this.responseText);
      txt += "<table border='1'>";
      for (x in myObj) {
        txt += "<tr><td>" + myObj[x].name + "</td></tr>";
      }
      txt += "</table>";
      document.getElementById("demo").innerHTML = txt;
    }
  };
  xmlhttp.open("POST", "json_demo_db_post.php", true);
```

```
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlhttp.send("x=" + dbParam);
}
</script>
```

https://www.w3schools.com/js/tryit.asp?filename=tryjson_html_table_dynamic

Domanda 21 (points: from 0 to 2) Scrivere il codice di un oggetto JSON che contenga i dati di una lista di persone a piacere (almeno due). Includere: nome, cognome, sesso, numeri di telefono (array), indirizzo (composto da via, comune, cap, provincia).

```
[  
  {  
    "nome": "Mario",  
    "cognome": "Rossi",  
    "sesso": "M",  
    "telefoni": ["089999999", "3333333333", "3344444444"],  
    "indirizzo": {  
      "via": "via Roma",  
      "comune": "Fisciano",  
      "cap": "84084",  
      "provincia": "SA"  
    }  
  },  
  {  
    "nome": "Carlo",  
    "cognome": "Verdi",  
    "sesso": "M",  
    "telefoni": ["089888888", "3311111111"],  
    "indirizzo": {  
      "via": "via Tirreno",  
      "comune": "Baronissi",  
      "cap": "84081",  
      "provincia": "SA"  
    }  
  }  
]
```

jQuery è una **libreria** JavaScript, che semplifica la programmazione JavaScript.

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide\

Lo scopo di **jQuery** è di rendere molto più semplice l'uso di JavaScript sul tuo sito web.

jQuery richiede molte attività comuni che richiedono molte linee di codice JavaScript e le racchiude in metodi che è possibile chiamare con una singola riga di codice, semplificando anche molte cose complicate da JavaScript, come le chiamate AJAX e la manipolazione DOM.

La libreria **jQuery** contiene le seguenti funzionalità:

- Manipolazione HTML / DOM
- Manipolazione CSS
- Metodi di eventi HTML
- Effetti e animazioni
- AJAX
- Utilità

Sintassi jQuery - https://www.w3schools.com/jquery/jquery_syntax.asp

La sintassi jQuery è fatta su misura per selezionare elementi HTML ed eseguire un'azione sugli elementi.

La sintassi di base è: **\$ (selettore).azione () :**

- Un segno \$ per accedere a **jQuery**
- Un (**selettore**) per "interrogare (o trovare)" elementi HTML
- Un **jQuery azione()** per eseguire sull'elemento (s)

\$(this).hide() - nasconde l'elemento corrente.

\$("p").hide() - nasconde tutti gli elementi <p>.

\$(".test").hide() - nasconde tutti gli elementi con class = "test".

\$("#test").hide() - nasconde l'elemento con id = "test".

L'evento ready():

```
$(document).ready(function(){  
    // jQuery methods go here...  
});
```

Questo per impedire l'esecuzione di qualsiasi codice jQuery prima che il documento abbia finito di caricare.

È buona norma aspettare che il documento sia completamente caricato e pronto prima di utilizzarlo. Ciò consente inoltre di avere il codice JavaScript prima del corpo del documento, nella sezione head.

Selettori - https://www.w3schools.com/jquery/jquery_selectors.asp

I selettori jQuery consentono di selezionare e manipolare gli elementi HTML. Sono utilizzati per "trovare" (o selezionare) elementi HTML in base al loro nome, id, classi, tipi, attributi, valori degli attributi e molto altro.

Tutti i selettori di jQuery iniziano con il simbolo del dollaro e parentesi: `$()`.

Selettore di elementi:

Il selettore di elementi jQuery seleziona gli elementi in base al nome dell'elemento: `$("p")`

Quando un utente fa clic su un pulsante, tutti gli elementi `<p>` saranno nascosti:

```
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_p

Selettore #id:

Il selettore jQuery utilizza l'attributo id di un tag HTML per trovare l'elemento specifico: `$("#test")`

```
$(document).ready(function(){
  $("button").click(function(){
    $("#test").hide();
  });
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_id

Selettore di classi:

Il selettore `.class` jQuery trova elementi con una classe specifica: `$(".test")`

Quando un utente fa clic su un pulsante, gli elementi con class = "test" saranno nascosti:

```
$(document).ready(function(){
  $("button").click(function(){
    $(".test").hide();
  });
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_class

The raw JavaScript way

I'm talking to the document
(aka the big D in DOM).

Get me all of the
elements that have
the tag name of "p."

document.getElementsByTagName("p")
[0].innerHTML = "Change the page..";

Get me the
zeroth element.

Set the HTML
inside that element... ...to this stuff.

The jQuery way

Grab me a
paragraph element.

\$("p").html ("Change the page..");

jQuery uses a "selector engine,"
which means you can get at stuff
with selectors just like CSS does.

Change the HTML of
that element to what's
in these parentheses.

Metodi evento - https://www.w3schools.com/jquery/jquery_events.asp

Un evento rappresenta il momento preciso in cui succede qualcosa.

\$(document).ready ():

Ci consente di eseguire una funzione quando il documento è completamente caricato.

```
$(document).ready(function(){
  // jQuery methods go here...
});
```

click():

La funzione viene eseguita quando l'utente fa clic sull'elemento HTML.

```
$( "p" ).click(function(){
  $(this).hide();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_click

dblClick ():

La funzione viene eseguita quando l'utente fa doppio clic sull'elemento HTML:

```
$( "p" ).dblclick(function(){
  $(this).hide();
});
```

<https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery dblclick>

mouseenter():

La funzione viene eseguita quando il puntatore del mouse entra nell'elemento HTML:

```
$("#p1").mouseenter(function(){
  alert("You entered p1!");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_mouseenter

mouseleave():

La funzione viene eseguita quando il puntatore del mouse lascia l'elemento HTML:

```
$("#p1").mouseleave(function(){
  alert("Bye! You now leave p1!");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_mouseleave

mousedown():

La funzione viene eseguita quando il pulsante sinistro, centrale o destro del mouse viene premuto, mentre il mouse si trova sopra l'elemento HTML:

```
$("#p1").mousedown(function(){
  alert("Mouse down over p1!");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_mousedown

mouseup():

La funzione viene eseguita quando viene rilasciato il pulsante sinistro, centrale o destro mentre il mouse si trova sopra l'elemento HTML:

```
$("#p1").mouseup(function(){
  alert("Mouse up over p1!");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_mouseup

hover():

La prima funzione viene eseguita quando il mouse entra nell'elemento HTML e la seconda funzione viene eseguita quando il mouse lascia l'elemento HTML:

```
$("#p1").hover(function(){
  alert("You entered p1!");
},
function(){
  alert("Bye! You now leave p1!");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hover

focus():

La funzione viene eseguita quando il campo modulo diventa attivo:

```
$( "input" ).focus(function(){
  $(this).css("background-color", "#cccccc");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_focus_blur

blur():

La funzione viene eseguita quando il campo modulo perde lo stato attivo:

```
$( "input" ).blur(function(){
  $(this).css("background-color", "#ffffff");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_focus_blur

on():

Il metodo **on()** allega uno o più gestori di eventi per gli elementi selezionati. Allegare un evento click a un elemento <p>:

```
$( "p" ).on("click", function(){
  $(this).hide();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_on_click

Effetti: nascondi e mostra - https://www.w3schools.com/jquery/jquery_hide_show.asp

Con jQuery, puoi nascondere e mostrare elementi HTML con i metodi **hide()** e **show()**:

```
$( "#hide" ).click(function(){
  $("p").hide();
});
$( "#show" ).click(function(){
  $("p").show();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_show

Sintassi:

```
$(selector).hide(speed,callback);
$(selector).show(speed,callback);
```

Il parametro di velocità opzionale specifica la velocità del nascondimento / visualizzazione e può assumere i seguenti valori: "low", "speed" o millisecondo.

Il parametro **callback** facoltativo è una funzione che deve essere eseguita dopo il completamento del metodo **hide()** o **show()**.

```
$( "button" ).click(function(){
  $("p").hide(1000);
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_slow

toggle ():

Puoi anche alternare tra nascondere e mostrare un elemento:

```
$( "button" ).click(function(){
  $( "p" ).toggle();
});
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_toggle

Sintassi:

```
$(selector).toggle(speed,callback);
```

Effetti jQuery – Dissolvenza - https://www.w3schools.com/jquery/jquery_fade.asp

Effetti jQuery – Scorrevole - https://www.w3schools.com/jquery/jquery_slide.asp

Effetti jQuery – Animazione - https://www.w3schools.com/jquery/jquery_animate.asp

jQuery Stop Animations - https://www.w3schools.com/jquery/jquery_stop.asp

Funzioni di callback - https://www.w3schools.com/jquery/jquery_callback.asp

Le istruzioni JavaScript vengono eseguite riga per riga. Tuttavia, con gli effetti, la prossima riga di codice può essere eseguita anche se l'effetto non è terminato. Questo può creare errori.

Per evitare ciò, è possibile creare una funzione di **callback**. Una funzione di **callback** viene eseguita dopo che l'effetto corrente è finito.

Sintassi tipica: **\$ (selettore) .hide (velocità, callback);**

L'esempio ha un parametro **callback** che è una funzione che verrà eseguita dopo che **hide** è stato completato:

```
$( "button" ).click(function(){
  $( "p" ).hide("slow", function(){
    alert("The paragraph is now hidden");
  });
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_hide_callback

jQuery – Chaining - https://www.w3schools.com/jquery/jquery_chaining.asp

Con jQuery, puoi concatenare azioni / metodi. Il concatenamento ci consente di eseguire più metodi jQuery (sullo stesso elemento) all'interno di una singola istruzione.

```
$( "#p1" ).css("color", "red").slideUp(2000).slideDown(2000);
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_chaining

Get su contenuti ed attributi - https://www.w3schools.com/jquery/jquery_dom_get.asp

Una parte molto importante di jQuery è la possibilità di manipolare il DOM. jQuery viene fornito con una serie di metodi correlati a DOM che semplificano l'accesso e la manipolazione di elementi e attributi.

Tre semplici, *ma utili*, metodi jQuery per la manipolazione DOM sono:

- **text()** - Imposta o restituisce il contenuto del testo degli elementi selezionati
- **html()** - Imposta o restituisce il contenuto degli elementi selezionati (incluso il markup HTML)
- **val()** - Imposta o restituisce il valore dei campi modulo

L'esempio seguente mostra come ottenere contenuti con metodi **text()** e **html()**:

```
$("#btn1").click(function(){
  alert("Text: " + $("#test").text());
});
$("#btn2").click(function(){
  alert("HTML: " + $("#test").html());
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_html_get

Nell'esempio seguente viene illustrato come ottenere il valore di un campo di input con il metodo **val()**:

```
$("#btn1").click(function(){
  alert("Value: " + $("#test").val());
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_val_get

Ottieni attributi - attr ():

Il metodo **attr()** viene utilizzato per ottenere valori di attributo.

Nell'esempio seguente viene illustrato come ottenere il valore dell'attributo **href** in un collegamento:

```
$("#button").click(function(){
  alert($("#w3s").attr("href"));
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_attr_get

Set su contenuti ed attributi - https://www.w3schools.com/jquery/jquery_dom_set.asp

L'esempio seguente mostra come impostare il contenuto con i metodi **text()**, **html()** e **val()**:

```
$("#btn1").click(function(){
  $("#test1").text("Hello world!");
});
$("#btn2").click(function(){
  $("#test2").html("<b>Hello world!</b>");
});
$("#btn3").click(function(){
  $("#test3").val("Dolly Duck");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_html_set

Imposta attributi - attr ():

Il metodo **attr()** viene anche utilizzato per impostare / modificare i valori degli attributi.

L'esempio seguente mostra come modificare (*impostare*) il valore dell'attributo href in un collegamento:

```
$("#button").click(function(){
  $("#w3s").attr("href", "https://www.w3schools.com/jquery/");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_attr_set

Add elementi - https://www.w3schools.com/jquery/jquery_dom_add.asp

Ci sono 4 metodi per aggiungere nuovi contenuti:

append():

Inserisce il contenuto ALLA FINE del contenuto degli elementi HTML selezionati (***Interno al tag***):

```
$( "p" ).append( "<p>Some appended text.</p>" );
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_html_append

prepend():

Inserisce il contenuto ALL'INIZIO del contenuto degli elementi HTML selezionati (***Interno al tag***):

```
$( "p" ).prepend( "<p>Some prepended text.</p>" );
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_html_prepending

```
function appendText() {  
    var txt1 = "<p>Text.</p>"; // Create element with HTML  
    var txt2 = $("<p></p>").text("Text."); // Create with jQuery  
    var txt3 = document.createElement("p"); // Create with DOM  
    txt3.innerHTML = "Text.";  
    $("body").append(txt1, txt2, txt3); // Append the new elements  
}
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_html_append2

after ():

inserisce il contenuto DOPO gli elementi HTML selezionati:

```
$( "img" ).after( "Some text after" );
```

before ():

inserisce il contenuto PRIMA degli elementi HTML selezionati:

```
$( "img" ).before( "Some text before" );
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_html_after

```
function afterText() {  
    var txt1 = "<b>I </b>"; // Create element with HTML  
    var txt2 = $("<i></i>").text("love "); // Create with jQuery  
    var txt3 = document.createElement("b"); // Create with DOM  
    txt3.innerHTML = "jQuery!";  
    $("img").after(txt1, txt2, txt3); // Insert new elements after <img>  
}
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_html_after2

Remove elementi - https://www.w3schools.com/jquery/jquery_dom_remove.asp

remove():

rimuove gli elementi selezionati e i relativi elementi figli:

```
$( "#div1" ).remove(); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_dom\_remove
```

Il metodo remove() accetta anche un parametro, che consente di filtrare gli elementi da rimuovere.

L'esempio seguente rimuove tutti gli <p>elementi con **class="test"**:

```
$( "p" ).remove( ".test" ); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_dom\_remove2
```

empty():

rimuove gli elementi secondari degli elementi selezionati:

```
$( "#div1" ).empty(); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_dom\_empty
```

Classi CSS - https://www.w3schools.com/jquery/jquery_css_classes.asp

addClass ():

Aggiunge una o più classi agli elementi selezionati:

```
$( "button" ).click(function(){  
    $("h1, h2, p").addClass("blue");  
    $("div").addClass("important");  
}); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_dom\_addclass
```

removeClass ():

Rimuove una o più classi dagli elementi selezionati:

```
$( "button" ).click( function() {
  $( "h1, h2, p" ).removeClass("blue");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_removeclass

toggleClass ():

Passa dall'aggiungere / rimuovere classi dagli elementi selezionati:

```
$( "button" ).click( function() {
  $( "h1, h2, p" ).toggleClass("blue");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_toggleclass

css() - https://www.w3schools.com/jquery/jquery_css.asp

Imposta o restituisce una o più proprietà di stile per gli elementi selezionati.

Per **restituire** il valore di una proprietà CSS specificata, utilizzare la sintassi: `css("propertynname");`

```
$( "p" ).css("background-color");
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_css_getcolor

Per **impostare** una proprietà CSS specificata, utilizzare la sintassi: `css("propertynname", "value");`

```
$( "p" ).css("background-color", "yellow");
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_css_setcolor

Dimensioni - https://www.w3schools.com/jquery/jquery_dimensions.asp

Il metodo **width()** imposta o restituisce la larghezza di un elemento (esclude padding, bordo e margine).

Il metodo **height()** imposta o restituisce l'altezza di un elemento (esclude padding, bordo e margine).

L'esempio seguente restituisce la larghezza e l'altezza di un elemento <div> specificato:

```
$( "button" ).click( function() {
  var txt = "";
  txt += "Width: " + $( "#div1" ).width() + "<br>";
  txt += "Height: " + $( "#div1" ).height();
  $( "#div1" ).html( txt );
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dim_width_height

Il metodo **innerWidth()** restituisce la larghezza di un elemento (include il padding).

Il metodo **innerHeight()** restituisce l'altezza di un elemento (include il padding).

L'esempio seguente restituisce la larghezza / altezza interna di un <div> elemento specificato :

```
$( "button" ).click( function() {
  var txt = "";
  txt += "Inner width: " + $( "#div1" ).innerWidth() + "<br>";
  txt += "Inner height: " + $( "#div1" ).innerHeight();
  $( "#div1" ).html( txt );
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dim_innerwidth_height

Il metodo **outerWidth()** restituisce la larghezza di un elemento (include padding e border).

Il metodo **outerHeight()** restituisce l'altezza di un elemento (include padding e border).

L'esempio seguente restituisce la larghezza / altezza esterna di un <div> elemento specificato :

```
$( "button" ).click( function(){
  var txt = "";
  txt += "Outer width: " + $( "#div1" ).outerWidth() + "<br>";
  txt += "Outer height: " + $( "#div1" ).outerHeight();
  $( "#div1" ).html( txt );
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dim_outerwidth_height

L'esempio seguente imposta la larghezza e l'altezza di un elemento <div> specificato :

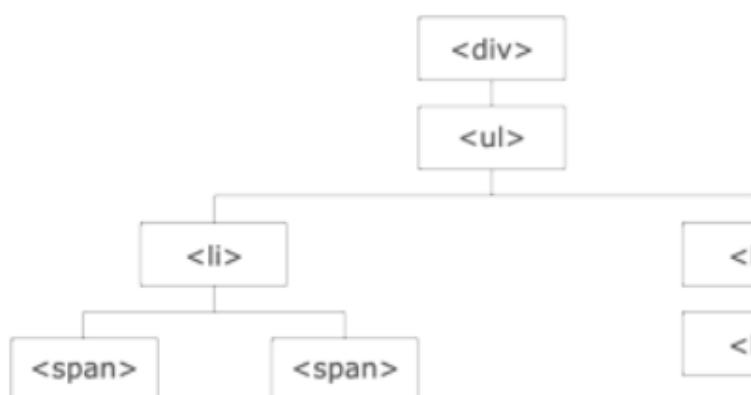
```
$( "button" ).click( function(){
  $( "#div1" ).width(500).height(500);
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dim_width_height_set

Traversing - https://www.w3schools.com/jquery/jquery_traversing.asp

Significa “Muoversi attraverso”, ed è usato per *trovare o selezionare* elementi HTML in base alla loro relazione con altri elementi. Inizia con una selezione e passa attraverso, fino a raggiungere gli elementi che desideri.

Con **jQuery traversing**, puoi facilmente muovere su (antenati), giù (discendenti) e lateralmente (fratelli) nell'albero, partendo dall'elemento (corrente) selezionato. Questo movimento è chiamato *attraversare o spostarsi* attraverso l'albero DOM. jQuery offre una varietà di metodi che ci permettono di attraversare il DOM.



La più grande categoria di metodi di attraversamento è il **tree-traversal**.

Ancestors - https://www.w3schools.com/jquery/jquery_traversing_ancestors.asp

parent ():

Restituisce l'elemento padre diretto dell'elemento selezionato. Questo metodo attraversa *solo un singolo* livello.

L'esempio seguente restituisce l'elemento padre diretto di ciascun elemento :

```
$(document).ready(function(){
  $("span").parent();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parent

parents ():

Restituisce tutti gli elementi degli antenati dell'elemento selezionato, fino all'elemento radice (<html>).

L'esempio seguente restituisce tutti gli antenati di tutti gli elementi :

```
$(document).ready(function(){
  $("span").parents();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parents

Puoi anche utilizzare un parametro facoltativo per filtrare la ricerca degli antenati.

L'esempio seguente restituisce tutti gli antenati di tutti gli elementi **** che sono elementi ****:

```
$(document).ready(function(){
  $("span").parents("ul");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parents2

parentsUntil ():

Restituisce tutti gli elementi antenati tra due argomenti dati.

L'esempio seguente restituisce tutti gli elementi antenati tra a **** e un elemento **<div>**:

```
$(document).ready(function(){
  $("span").parentsUntil("div");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parentsuntil

Descendants - https://www.w3schools.com/jquery/jquery_traversing_descendants.asp

children ():

Restituisce tutti i figli diretti dell'elemento selezionato. Questo metodo attraversa solo un singolo livello lungo l'albero DOM.

L'esempio seguente restituisce tutti gli elementi che sono figli diretti di ciascun elemento **<div>**:

```
$(document).ready(function(){
  $("div").children();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_children

Puoi anche utilizzare un parametro facoltativo per filtrare la ricerca di bambini.

L'esempio seguente restituisce tutti gli **<p>** elementi con il nome della classe "**first**", che sono figli diretti di **<div>**:

```
$(document).ready(function(){
  $("div").children("p.first");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_children2

find ():

Restituisce elementi discendenti dell'elemento selezionato, fino all'ultimo discendente.

L'esempio seguente restituisce tutti gli elementi **** che sono discendenti di **<div>**:

```
$(document).ready(function(){
  $("div").find("span");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_find

L'esempio seguente restituisce tutti i discendenti di **<div>**:

```
$(document).ready(function(){
  $("div").find("*");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_find2

Siblings (fratelli) - https://www.w3schools.com/jquery/jquery_traversing_siblings.asp

siblings():

Restituisce tutti gli elementi di pari livello dell'elemento selezionato.

L'esempio seguente restituisce tutti gli elementi di pari livello di **<h2>**:

```
$(document).ready(function(){
  $("h2").siblings();
});
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_siblings
```

Puoi anche usare un parametro opzionale per filtrare la ricerca dei fratelli.

Nell'esempio seguente vengono restituiti tutti gli elementi di pari livello di **<h2>** che sono elementi **<p>**:

```
$(document).ready(function(){
  $("h2").siblings("p");
});
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_siblings2
```

next ():

Restituisce il prossimo elemento fratello dell'elemento selezionato.

L'esempio seguente restituisce il fratello successivo di **<h2>**:

```
$(document).ready(function(){
  $("h2").next();
});
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_next
```

nextAll ():

Restituisce tutti i successivi elementi di pari livello dell'elemento selezionato.

L'esempio seguente restituisce tutti i successivi elementi di pari livello di **<h2>**:

```
$(document).ready(function(){
  $("h2").nextAll();
});
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_nextall
```

nextUntil ():

Restituisce tutti i successivi elementi di pari livello tra due argomenti dati.

L'esempio seguente restituisce tutti gli elementi di pari livello tra a **<h2>** e un elemento **<h6>**:

```
$(document).ready(function(){
  $("h2").nextUntil("h6");
});
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_nextuntil
```

I metodi **prev()**, **prevAll()** e **prevUntil()** funzionano come i metodi di cui sopra, ma con funzionalità inverso: tornano elementi di pari livello precedenti (attraversano indietro lungo elementi di pari livello nell'albero DOM, invece che in avanti).

Filtering - https://www.w3schools.com/jquery/jquery_traversing_filtering.asp

first ():

Restituisce il primo elemento degli elementi specificati.

L'esempio seguente seleziona il primo elemento **<div>**:

```
$(document).ready(function(){
  $("div").first();
});
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_first
```

last ():

Restituisce l'ultimo elemento degli elementi specificati.

L'esempio seguente seleziona l'ultimo elemento **<div>**:

```
$(document).ready(function(){
  $("div").last();
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_last

eq ():

Restituisce un elemento con un numero indice specifico degli elementi selezionati.

I numeri indice iniziano da 0, quindi il primo elemento avrà il numero indice 0 e non 1. L'esempio seguente seleziona il secondo elemento <p> (numero indice 1):

```
$(document).ready(function(){
  $("p").eq(1);
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_eq

filter ():

Consente di specificare un criterio. Gli elementi che non corrispondono ai criteri vengono rimossi dalla selezione e quelli che corrispondono saranno restituiti.

L'esempio seguente restituisce tutti gli elementi <p> con nome di classe "**intro**":

```
$(document).ready(function(){
  $("p").filter(".intro");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_filter

not ():

Restituisce tutti gli elementi che non corrispondono ai criteri.

Suggerimento: il metodo **not()** è l'opposto di **filter()**.

L'esempio seguente restituisce tutti gli elementi <p> che non hanno il nome di classe "**intro**":

```
$(document).ready(function(){
  $("p").not(".intro");
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_not

jQuery - Introduzione AJAX - https://www.w3schools.com/jquery/jquery_ajax_intro.asp

AJAX è l'arte di scambiare dati con un server e di aggiornare parti di una pagina web senza ricaricare l'intera pagina.

AJAX = JavaScript asincrono e XML. AJAX riguarda il caricamento dei dati in background e la loro visualizzazione.

Con i metodi **jQuery AJAX**, puoi richiedere **testo**, **HTML**, **XML** o **JSON** da un server remoto utilizzando **HTTP Get** e **HTTP Post**. E puoi caricare i dati esterni direttamente negli elementi HTML selezionati della tua pagina web.

jQuery - Metodo AJAX load() - https://www.w3schools.com/jquery/jquery_ajax_load.asp

Il metodo **load()** carica i dati da un server e inserisce i dati restituiti nell'elemento selezionato.

Sintassi: `$(selector).load(URL,data,callback);`

- Il parametro **URL** richiesto specifica l'URL che desideri caricare.
- Il parametro **facoltativo data** specifica un insieme di coppie chiave / valore **querystring** da inviare insieme alla richiesta.
- Il parametro **facoltativo callback** è il nome di una funzione da eseguire dopo il completamento del metodo **load()**.

Contenuto del file .txt :

```
<h2>jQuery and AJAX is FUN!!!</h2>
<p id="p1">This is some text in a paragraph.</p>
```

L'esempio seguente carica il contenuto del file "**demo_test.txt**" in un elemento **<div>** specifico :

```
$("#div1").load("demo_test.txt"); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_ajax\_load
```

È anche possibile aggiungere un **selettore jQuery** al parametro **URL**.

L'esempio seguente carica il contenuto dell'elemento con **id = "p1"**, all'interno del file "**demo_test.txt**", in un elemento **<div>** specifico :

```
$("#div1").load("demo_test.txt #p1");
https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_ajax\_load2
```

La funzione di **callback** può avere parametri diversi:

- **responseTxt** - contiene il **contenuto risultante** se la chiamata ha esito positivo
- **statusTxt** - contiene lo **stato** della chiamata
- **xhr** - contiene **l'oggetto XMLHttpRequest**

L'esempio seguente mostra una finestra di avviso dopo il completamento del metodo **load ()**, se ha avuto successo, visualizza "Contenuto esterno caricato con successo!", E se fallisce visualizza un messaggio di errore:

```
$(button).click(function(){
  $("#div1").load("demo_test.txt", function(responseTxt, statusTxt, xhr){
    if(statusTxt == "success")
      alert("External content loaded successfully!");
    if(statusTxt == "error")
      alert("Error: " + xhr.status + ": " + xhr.statusText);
  });
}); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_ajax\_load\_callback
```

jQuery metodi AJAX get() post() - https://www.w3schools.com/jquery/jquery_ajax_get_post.asp

I metodi **jQuery get()** e **post()** vengono utilizzati per richiedere dati dal server con una richiesta **HTTP GET** o **POST**.

GET è fondamentalmente utilizzato per ottenere (recuperare) alcuni dati dal server. **Nota:** il metodo **GET** può restituire dati memorizzati nella cache.

Il **POST** può anche essere usato per ottenere alcuni dati dal server. Tuttavia, il metodo **POST** non memorizza **MAI** dati e viene spesso utilizzato per inviare dati insieme alla richiesta.

Metodo jQuery **\$.get()**:

Sintassi: **\$.get(URL,callback);**

- Il parametro **URL** richiesto specifica l'URL che desideri richiedere.
- Il parametro **facoltativo callback** è il nome di una funzione da eseguire se la richiesta ha esito positivo.

Nell'esempio seguente viene utilizzato il metodo **\$.get()** per recuperare i dati da un file sul server:

```
$(button).click(function(){
  $.get("demo_test.asp", function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
}); https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery\_ajax\_get
```

Il primo parametro di **callback** contiene il contenuto della pagina richiesta e il secondo parametro di **callback** mantiene lo stato della richiesta.

demo_test.asp:

```
<%
response.write("This is some text from an external ASP file.")
%>
```

Metodo jQuery \$.post():

Sintassi: **\$.post(URL,data,callback);**

- Il parametro **URL** richiesto specifica l'URL che desideri richiedere.
- Il parametro *facoltativo dati* specifica alcuni dati da inviare insieme alla richiesta.
- Il parametro *facoltativo callback* è il nome di una funzione da eseguire se la richiesta ha esito positivo.

Nell'esempio seguente viene utilizzato il metodo **\$.post()** per inviare alcuni dati insieme alla richiesta:

```
$( "button" ).click( function(){
  $.post( "demo_test_post.asp",
  {
    name: "Donald Duck",
    city: "Duckburg"
  },
  function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_post

Lo script ASP in "**demo_test_post.asp**" legge i parametri, li elabora e restituisce un risultato.

Il primo parametro di **callback** contiene il contenuto della pagina richiesta e il secondo parametro di **callback** mantiene lo stato della richiesta.

demo_test_post.asp:

```
<%
dim fname,city
fname=Request.Form("name")
city=Request.Form("city")
Response.Write("Dear " & fname & ". ")
Response.Write("Hope you live well in " & city & ".")
%>
```

jQuery – Filters per ricerche - https://www.w3schools.com/jquery/jquery_filters.asp