UniCT – DMI

Web Programming and Usability

ANNO ACCADEMICO

2020/21

Autori:

Alessio Tudisco

Web Programming

Introduzione

Cos'è HTTP?

Il protocollo *HTTP* è un protocollo *stateless* (il server non deve conservare informazioni o uno stato per ogni client che effettua una comunicazione) del *livello applicazione*, che permette la comunicazione tra *sistemi distribuiti,* ovvero la comunicazione fra *vari host e vari client*. In pratica: *il client invia al server una richiesta HTTP richiedendo una risorsa e il server risponde attraverso una risposta HTTP*.

Le richieste HTTP, sulle quali si basa la comunicazione web, sono inviate mediante gli *URL (Uniform Resource Locators).* Un URL è caratterizzato dalla seguente struttura:

$$[\boldsymbol{PROTOCOL}]://[\boldsymbol{DOMAIN}]:[\boldsymbol{PORT}]/[\boldsymbol{RESOURCE}]?[\boldsymbol{QUERY}]$$

Come funzionano le richieste HTTP?

L'URL *identifica la risorsa*, ma l'azione che si vuole far eseguire all'host viene indicata attraverso i *verbi HTTP:*

- *GET* per la *richiesta* di una risorsa esistente;
- *POST* per la *creazione* di una nuova risorsa;
- *PUT* per l'*aggiornamento* di risorsa esistente;
- *DELETE* per la *rimozione* di una risorsa esistente;

I verbi *PUT* e *DELETE* sono spesso considerati una specializzazione di *POST*.

Comparazione fra GET e POST

GET

Il metodo GET manda le informazioni in chiaro mettendole nella sezione *QUERY* dell'URL.

Le specifiche del protocollo HTTP, indicano che nel caso di una richiesta GET nessun dato dovrebbe essere scritto nel body della richiesta.

L'utilizzo del metodo GET comporta i seguenti benefici/limitazioni:

- Le richieste GET *possono essere cachate*;
- Le richieste GET *possono creare una cronologia nel browser*;
- Le richieste GET *possono essere salvate nei preferiti* poiché i parametri sono presenti in chiaro nell'URL;
    - Per tale motivo è sconsigliato usarle quando si ha a che fare con dati sensibili;
- Le richieste GET *sono dipendenti dai limiti di lunghezza degli URL* dei webserver e degli applicativi;
    - Per esempio Internet Explorer supportava URL fino a 2048 caratteri e vecchie versioni di Apache supportavano URL fino a 4096 caratteri;
- Le richieste GET *supportano solo parametri di tipo testuale*;
    - Si possono comunque inviare altri tipi di dati se viene prima effettuata una codifica base64 o simile;

POST

Il metodo POST manda le informazioni *ponendole nel body della richiesta HTTP*.

L'utilizzo del metodo POST comporta i seguenti benefici/limitazioni:

- Le richieste POST *non possono essere cachate*;
- Le richieste POST *non possono creare una cronologia nel browser*;
- Le richieste POST *non possono essere salvate nei preferiti*;
    - Per tale motivo è consigliato usarle quando si ha a che fare con dati sensibili;
- Le richieste *POST non dipendono dai limiti di lunghezza degli URL* dei webserver e degli applicativi;

- Le richieste POST *supportano parametri di tipo numerico, testuale, binario, ect*…

Gli status code delle risposte HTTP

Attraverso un URL ed un verbo, un client può inizializzare una richiesta HTTP ad un server. In risposta, il server fornirà un messaggio ed uno status code (*codice di stato*). I codici sono categorizzati come segue:

- *1XX*: per i messaggi informativi;
- *2XX*: per i messaggi di operazione riuscita;
- *3XX*: per i messaggi di avvenuto caching;
- *4XX*: per i messaggi di errore lato client;
- *5XX*: per i messaggi di errore lato server;

HTML

L'HTML (*Hyper-Text Markup Language*) è *il linguaggio di markup* standard per la creazione di pagine web, esso rappresenta la struttura di una pagina web attraverso dei *<tag>.*

Il ruolo principale di un web browser è quello di leggere un documento HTML e visualizzarne il contenuto. Così come succede per i file eseguibili con il *magic number*, i documenti HTML prevedono un *tag iniziale* che aiuta i web browser a visualizzare correttamente i siti web.

Questo tag prende il nome di *DOCTYPE* e a seconda della versione di HTML che si vuole usare ha la seguente forma:

- Per html 4.01:
  $< !DOCTYPE\ HTML\ PUBLIC\ "-//W3C//DTD\ HTML\ 4.01//EN"\ "http://www.w3.org/TR/html4/strict.dtd" >$
- Per html 5:

$$< !DOCTYPE\ html >$$

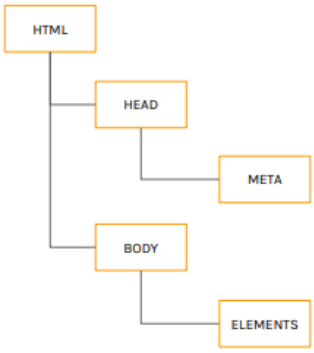
Il linguaggio *HTML* deriva dal *meta-linguaggio XML* e appartiene alla famiglia degli *Standard Generalized Markup Language (SGML)*, è quindi un linguaggio *case-insensitive* caratterizzato dall'uso di *tag,* che verranno spiegate a breve.

La struttura standard di un file HTML

Come abbiamo già detto, la parte iniziale di un file HTML è costituita dal tag speciale *DOCTYPE*. Subito dopo troviamo una struttura pressoché standard: vi è un *tag contenitore* principale denominato *<html>* che contiene 2 parti fondamentali:

- Head: è una sezione dedicata ai metadati od altre informazioni che non contribuiscono visivamente alla pagina web, come ad esempio il titolo della scheda del browser o i metadati di OpenGraph;
- Body: è la sezione dedicata agli elementi che contribuiscono visivamente alla pagina web, ovvero contiene tutti gli elementi che l'utente vedrà e con cui potrà interagire.

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.     <title>Title here!</title>
5.     <!-- META INFO -->
6.  </head>
7.
8.  <body>
9.
10.    <!-- ELEMENTS HERE -->
11.
12. </body>
13. </html>
```



Una struttura HTML è definita da un albero n-ario, in cui tutti i nodi sono elementi HTML. Gli elementi HTML possono essere annidati, ovvero possono contenere altri elementi HTML.
Un elemento HTML inizia con un *tag di apertura, e, se contengono qualcosa, un tag di chiusura:*

- *<Pippo>:* apertura di un tag dell'elemento *Pippo;*
- *</Pippo>:* chiusura di un tag dell'elemento *Pippo;*

$$< \boldsymbol{Pippo} > \ \boldsymbol{Contenuto}\ < /\boldsymbol{Pippo} >$$

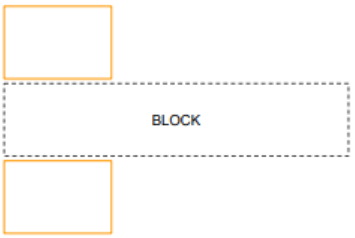
Alcuni elementi HTML denominati *void elements* non necessitano di tag di chiusura poiché non prevedono alcun contenuto e presentano un *forward-slash* nel tag di apertura:

$$< \boldsymbol{br}/> \ | \ < \boldsymbol{hr}/> \ | \ < \boldsymbol{input}/> \ | \ < \boldsymbol{img}/>$$

Elementi block ed inline

Gli elementi HTML possono essere categorizzati in due categorie a seconda di alcuni comportamenti particolari.

Definiamo *elementi di tipo* block, gli elementi che:



- Iniziano sempre a capo;
- Prendono tutta la larghezza disponibile;
- Hanno margini superiori e inferiori;

Definiamo *elementi di tipo* inline, gli elementi che:

- Non iniziano a capo;
- Prendono solo la larghezza necessaria per il proprio contenuto;
- Non hanno margini superiori e inferiori;

Attributi degli Elementi

Gli attributi sono utilizzati per le caratteristiche di un elemento HTML. Gli attributi vengono inseriti all'interno del *tag di apertura* dell'elemento HTML ed hanno una forma di tipo *chiave-valore*.

$$\boldsymbol{ATTRIBUTE = "VALUE"}$$

In termini di OOP, è paragonabile agli attributi di un oggetto.

Fra i tanti attributi utilizzabili esistono *4 attributi nativi*, ovvero che possono essere utilizzati in *qualunque* elemento HTML:

- ID: un identificatore univoco dell'elemento;
- CLASS: un identificatore di relazione che raggruppa più elementi HTML;
- STYLE: permette di definire uno stile per l'elemento HTML;
- TITLE: una descrizione che appare attraverso un tooltip, spesso utilizzata pure dai *narratori per fornisce maggiore accessibilità agli utenti non vedenti*.

Inoltre è possibile utilizzare dei *meta-attributi*, ovvero attributo *non standard del HTML ma custom*, che possono contenere informazioni utili per il corretto funzionamento della pagina web. La convenzione generale per i *meta-attributi* è la seguente: (ovvero l'utilizzo del prefisso *data-*)

$$\boldsymbol{data - META\_ATTRIBUTE = "VALUE"}$$

Commenti all'interno di un documento HTML

Il commento è un pezzo di codice che viene ignorato dai browser web durante la fase di costruzione della pagina web. È una buona pratica aggiungere commenti nel codice HTML, specialmente in documenti complessi, per indicare sezioni di un documento e qualsiasi altra nota a chiunque guardi il codice.

Nel linguaggio HTML i commenti sono marcati dall'uso dell'operatore *freccia*:

```
<!-- COMMENT HERE  -->
```

Alcuni elementi HTML

Heading

Gli *heading* sono *elementi HTML di tipo block* che rappresentano *titoli o sottotitoli* che si vogliono visualizzare nella pagina web. Il linguaggio HTML prevede *6 livelli di heading* per *indicare l'importanza nel documento*: si va dall'1, il più grande e in grassetto, fino al 6, *man mano più piccolo con un grassetto man mano più lieve.*

```
<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 è: $< \boldsymbol{hX} > \ \boldsymbol{contenuto} \ < /\boldsymbol{hX} >$, *dove a X si sostituisce il livello.*

Paragrafi

I *paragrafi* sono *elementi HTML di tipo block* che rappresentano *blocchi di testo*. Per convenzione i browser aggiungono un margine prima e dopo l'elemento paragrafo. Il *tag* è come segue:

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

Formattazione dei testi

Sono presenti numerosi *elementi HTML di tipo inline* che possono essere usati per definire una *formattazione* del testo. I principali sono:

- *<b>:* conferisce al testo che contiene l'effetto *grassetto*;
- **: conferisce al testo che contiene l'effetto *grassetto* e indica *un livello di importanza superiore*, questo elemento HTML è sfruttato dai *search engine* per eseguire l'indexing delle pagine web;
- *<i>:* conferisce al testo che contiene l'effetto *corsivo*;
- **: conferisce al testo che contiene l'effetto *corsivo* e indica *un livello di importanza superiore*, questo elemento HTML è sfruttato dai *search engine* per eseguire l'indexing delle pagine web;
- *<mark>:* conferisce al testo che contiene un effetto di *sottolineatura di evidenziatore*;
- *<small>*: conferisce al testo che contiene una *dimensione ridotta* rispetto alla dimensione standard;
- *<del>*: conferisce al testo che contiene un effetto *barrato;*
- *<ins>*: conferisce al testo che contiene un effetto di *sottolineatura*;
- *<sub>*: conferisce al testo che contiene una dimensione e una posizione simile a quella di un *pedice matematico*;
- *<sup>*: conferisce al testo che contiene una dimensione e una posizione simile a quella di un *esponente matematico*;

Link ipertestuali (hyperlink)

Gli hyperlink sono *elementi HTML di tipo inline* che permettono ai visitatori di *navigare tra siti web* cliccando su parole, frasi o immagini. Il *tag* è come segue:

```
<a href="https://www.w3schools.com/html/" title="Go to W3Schools HTML section" target="_black">Visit our HTML Tutorial</a>
```

Gli *attributi specifici* di un hyperlink sono i seguenti:

- *href:* definisce l'url della destinazione;
- *target:* definisce la modalità in cui verrà effettuata la navigazione:
  - *_self: la nuova pagina verrà aperta sulla tab corrente del browser;*
  - *_blank: la nuova pagina verrà aperta su una nuova tab del browser;*

Ancore (anchors)

Gli hyperlink possono essere utilizzati per implementare *una navigazione interna* nella pagina web che permette all'utente di effettuare un *salto* verso l'informazione desiderata. Gli hyperlink che effettuato la navigazione interna prendono il nome di *ancore*.

La creazione di un ancora è assai semplice: basta inserire come attributo *href* l'ID dell*'elemento HTML su cui si vuole effettuare il salto*.

Uno degli utilizzi più comune delle ancore è la *tabella dei contenuti (TOC).*

```
<p><a href="#C4">Jump to Chapter 4</a></p>
<p><a href="#C10">Jump to Chapter 10</a></p>
...
<h2 id="C4">Chapter 4</h2>
<p>This chapter explains ba bla bla</p>
...
<h2 id="C10">Chapter 10</h2>
<p>This chapter explains ba bla bla</p>
```

Immagini

In HTML la visualizzazione di un'immagine avviene attraverso un *elemento HTML void di tipo inline.* Il *tag* è come segue:

```
<img src="pic_trulli.jpg" alt="Italian Trulli">
```

Gli *attributi specifici* di un'immagine sono i seguenti:

- *src:* definisce l'url in ci si trova l'immagine oppure la sua codifica in base64;
- *alt:* il testo che viene visualizzato quando l'immagine deve essere ancora reperita o quando l'immagine non è stata trovata. *Questo tag viene inoltre utilizzato dai narratori per gli utenti non vedenti;*
- *width:* definisce un ridimensionamento forzato della lunghezza dell'immagine;
- *height:* definisce un ridimensionamento forzato dell'altezza dell'immagine;

Liste (Elenchi puntati)

Le liste sono *elementi HTML di tipo block* che permettono di visualizzare un elenco formato da vari elementi. Vi è un *elemento contenitore* che contiene gli elementi **<li>** delle singole voci listate*, entrambi gli elementi sono di tipo block.* Il *tag contenitore* cambia a seconda del *tipo di lista* che vogliamo visualizzare:

- Una lista *ordinata (Ordered List),* ovvero con una numerazione degli elementi, ha come *tag* ***<ol>****;*
- Una lista non *ordinata (Unordered List),* ovvero senza una numerazione degli elementi, ha come *tag* ***<ul>****;*

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

- Coffee
- Tea
- Milk

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

1. Coffee
2. Tea
3. Milk

Tabelle

Le tabelle sono *elementi HTML di tipo block* che permettono di rappresentare dati tabulari, ma furono adottate in principio anche per strutturare graficamente le pagine web.

La struttura di una tabella HTML è abbastanza articolata, è formata da più elementi aventi un ruolo specifico, ed è definita riga per riga:

- ***<table>:*** costituisce il *tag contenitore principale* della tabella, contiene la definizione di *ogni riga* della tabella;
    - A tale tag è possibile definire l'attributo *border, avente valore numerico,* per visualizzare uno stile di bordo;
- ***<tr>:*** costituisce *una singola riga della tabella*, contiene gli elementi contenente i dati o le intestazioni;
- ***<th>:*** costituisce una cella di testo (o altro) che ha il ruolo di *intestazione*, tale testo presente un effetto *grassetto*;
- ***<td>:*** costituisce una cella di testo (o altro) che rappresenta il dato che si vuole tabulare.

```
<table border=1>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td rowspan=2>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
  </tr>
  <tr>
    <td>John</td>
    <td>Doe</td>
    <td>80</td>
  </tr>
</table>
```

Alle *celle* è possibile definire *attributi specifici:*

- *colspan:* definisce un comportamento di *espansione* della cella su altre celle della stessa riga;
- *rowspan:* definisce un comportamento di *espansione* della cella su altre celle della stessa colonna;

Divisore (DIV)

Il *divisore* è un *elemento HTML di tipo block* utilizzato per rappresentare *sezione* del documento HTML, infatti *divide* il contenuto della pagina web.

È canonicamente utilizzato come contenitore di altri elementi HTML appartenenti logicamente alla stessa sezione.

```
<div class="navbar">
  ...
</div>
<div class="article">
  ...
</div>
```

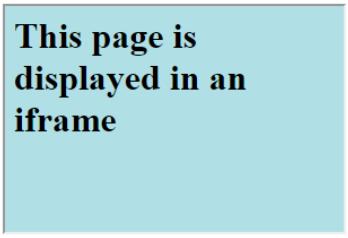
Iframe

L'iframe è un *elemento HTML di tipo inline* che permette di visualizzare una pagina web all'interno di un'altra pagina web.

```
<iframe src="url" title="description"></iframe>
```

Gli *attributi specifici* di un *iframe* sono i seguenti:

- *src: l'indirizzo url* della pagina web che si vuole "importare";
- *title:* titolo testuale usato dai narratori;
- *width:* definisce la lunghezza della finestra di visualizzazione;
- *height:* definisce l'altezza della finestra di visualizzazione;



Button

Un bottone è un *elemento HTML di tipo inline* che rappresenta un elemento cliccabile, può contenere testo, immagini ed elementi HTML di formattazione del testo. Il *tag* è come segue:

```
<button type="button">Click Me!</button>
```

I *principali attributi specifici* di un *button* sono:

- *type:* per indicare al web browser la tipologia di bottone;
- *form:* specifica l'id dell'elemento HTML *form* a cui esso appartiene;

Form e Input

Il form è *un elemento HTML di tipo block* **interattivo** con cui l'utente interagisce al fine di inviarci delle informazioni. Il form è un *elemento contenitore* di *elementi input, elementi HTML di tipo inline.*

Gli attributi specifici di un *form* sono i seguenti:

- *action*: definisce l'azione da performare al submit, generalmente è l'indirizzo di una pagina web o script su un server;
- *method*: specifica il verbo http da utilizzare al submit, un form può eseguire GET o POST;

```
<form action="url" method="GET">
.
form elements
.
</form>
```

Un input è un *elemento HTML void di tipo inline* che può raccogliere un dato di una particolare forma, può assume le sembianze di una: textfield, checkbox, radio button, combobox, etc. Il tag è il seguente:

$$< \boldsymbol{input\ type} = "\{\boldsymbol{TYPES}\}"\ \boldsymbol{name} = "\{\boldsymbol{NAME\ OF\ KEY}\}"\ [\boldsymbol{OTHER\ ATTRIBUTES}]\ />$$

Gli *attributi specifici fondamentali* di un input sono i seguenti:

- *type:* definisce la *tipologia* di dato che può raccogliere, definisce perciò le *sembianze* e la *funzionalità* dell'elemento;
- *name:* definisce il *nome* dell'input, ovvero *l'identificativo* con cui il dato viene inviato;
  - *Nel caso in cui l'attributo name viene omesso, il dato che esso ha raccolta non verrà inviato;*

Tipi di <input>

- ***<input type="button">***: un bottone cliccabile, si differenzia dall'elemento *HTML* ***<button>*** poiché quest'ultimo può contenere testo, immagini o *elementi HTML di formattazione del testo;*
- ***<input type="checkbox">***: un quadratino selezionabile per asserire un'opzione con risultato *vero o falso*;
- ***<input type="color">***: un color picker per permettere all'utente la selezione di un colore;
- ***<input type="date">***: un calendario che permette all'utente di selezionare una data;
- ***<input type="email">***: un textfield a singola riga specializzato alla raccolta di un indirizzo email;
- ***<input type="file">***: un selettore di file per un eventuale upload;
- ***<input type="hidden">***: un elemento nascosto che viene usato per contenere informazioni che verranno inviate alle pagine web o script;
- ***<input type="number">***: un selettore numerico;
- ***<input type="password">***: un textfield a singola riga specializzato alla raccolta di una password, il testo è sostituito da un carattere speciale come l'asterisco;
- ***<input type="radio">***: un cerchio cliccabile usato per la selezione di un'opzione. *A differenza di un checkbox, vengono definiti più input radio con* ***lo stesso nome*** *e* ***solo l'opzione selezionata*** *verrà inviata;*
- ***<input type="range">***: uno slider che permette all'utente di selezionare un numero fra un range predefinito;
- ***<input type="reset">***: un bottone che permette la pulizia di tutti gli input del form;
- ***<input type="submit">***: un bottone che ha lo scopo di "lanciare" l'azione del form;
- ***<input type="tel">***: un textfield a singola riga specializzato alla raccolta di un numero telefonico;
- ***<input type="text">***: un textfield in cui l'utente può scrivere una singola riga di testo;
- ***<input type="time">***: un time picker, permette all'utente di selezionare un determinato orario;
- ***<input type="url">***: un textfield a singola riga specializzato alla raccolta di un URL;

Gli *input* dispongono dei seguenti *attributi opzionali o di comportamento*:

- *value*: specifica il valore iniziale dell'input;

- *placeholder:* specifica un suggerimento su che tipo di valore inserire;
- *readonly*: specifica che l'input non è modificabile;
- *disabled*: specifica che l'input è disabilitato e non verrà incluso nella richiesta HTTP;
- *required*: specifica che l'input è necessario per poter inviare la richiesta HTTP, se non viene compilato tale input al submit si otterrà un avviso;
- *checked*: specifica che l'input di tipo checkbox o radio è selezionato in modo predefinito;
- *pattern*: specifica un regex con il quale l'input può validare il dato che raccoglie;
- *maxlength*: specifica la massima lunghezza del dato raccoglibile dall'input;
- *min* e *max*: specificano il valore minimo e quello massimo accettabili dall'input, può essere un numero o una data;

TextArea

La *textarea* è un *elemento HTML di tipo inline* che consiste in un *textfield multi linea.* Tale input ha un tag proprio ed è il seguente:

```
<textarea id="w3review" name="w3review" rows="4" cols="50">
  Text Text Text
</textarea>
```

Gli *attributi specifici* di una *textarea* sono:

- *rows*: specifica da quante righe è formata;
- *cols:* specifica da quante colonne è formata;

Select (ComboBox)

Il *select* è un *elemento HTML di tipo inline* che permette la visualizzazione di un menù di opzioni e la scelta di una o più di esse.

La struttura di un *select* è abbastanza semplice, vi è:

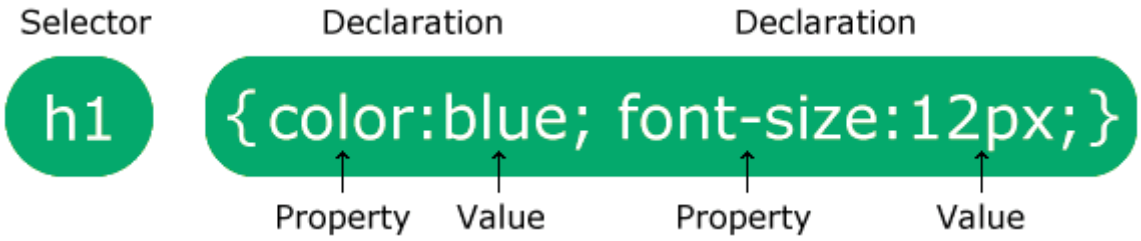
```
<select name="cars" id="cars" multiple>
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
</select>
```

- ***<select>***: è il *tag contenitore* che contiene le singole opzioni;
  - È possibile specificare l'attributo **multiple** *per permettere la selezione multipla;*
- ***<option>***: è il tag che contiene una singola opzione;
  - L'attributo value specifica l'identificativo dell'opzione;

CSS

CSS sta per *Cascading Style Sheet* ed è un *linguaggio di stile* che permette di descrivere il layout delle pagine web. Consiste in una serie di regole di stile, aventi forma *chiave-valore*, che vengono applicate a cascata, ovvero dall'alto verso il basso in cui: se una regola viene definita più volte, prevale la sua ultima definizione.

Vengono utilizzati dei *selettori* per individuare gli elementi dell'*albero HTML* a cui applicare le regole contenute nel *declaration block:*



Per una migliore comprensione di un selettore CSS è buona norma leggere il selettore da destra verso sinistra!

Tipologie di selettori CSS

I selettori CSS sono utilizzati per individuare gli elementi HTML nella pagina web a cui vogliamo aggiungere regole di stile. Esistono vari tipi di selettori, ognuno con una propria funzionalità.

Selettori semplici

Element selector

Gli *element selectors* sono i selettori più semplici: eseguono un match **case-insensitive** tra il nome del selettore ed il tag HTML:

In questo esempio, il selettore *p* individua tutti i *paragrafi (si ricorda che il tag di un paragrafo è **<p>**)* del documento HTML ed applica gli stili di allineamento e colore del testo.

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

ID selectors

Gli *id selectors* consistono di un "**#**" seguito dall'ID di un elemento HTML a cui applicare le regole di stile. Supponendo che ogni elemento abbia un ID univoco, lo stile verrà applicato a quell'unico elemento.

In questo esempio, il selettore "#para1" individua *l'unico elemento HTML avente come **para1*** come ID ed applica gli stili di allineamento e colore del testo.

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

Class selectors

I *class selectors* consistono di un "**.**" seguito dal nome della classe sotto il cui raggruppare tutti gli elementi a cui applicare le regole di stile. Più elementi di vario tipo in una struttura HTML possono avere la stessa classe ed un elemento può avere più classi.

In questo esempio, il selettore "*.center*" individua tutti gli *elementi HTML facenti appartenenza alla classe **center*** ed applica gli stili di allineamento e colore del testo.

```
.center {
  text-align: center;
  color: red;
}
```

Universal selector

Il *selettore universale* è costituito da un (*) ed applica le regole di stile a tutti gli elementi della struttura HTML. *Se non utilizzato con attenzione, può impattare molto sulle performance del rendering della pagina.* Può *risultare più utile* se usato **in combinazione** con gli altri selettori.

In questo esempio, il selettore universale *applica a tutti gli elementi HTML* gli stili di allineamento e colore del testo.

```
* {
  text-align: center;
  color: blue;
}
```

Combinazione di selettori

È possibile *combinare* più selettori CSS tra loro per ottenere dei particolari *pattern di individuazione* degli *elementi HTML.* Vi sono *svariate combinazioni con effetti diversi*.

Raggruppamento (selettori separati da una virgola)

È possibile usare una combinazione di *selettori separati da una virgola* come *selettore* unico di una serie di regole di stile per applicare tali regole a tutti i singoli selettori della combinazione.

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

In questo esempio, questa combinazione individua *tutti gli elementi HTML di tipo h1, h2 e p* ed applica gli stili di allineamento e colore del testo.

Descendant selector (selettori separati da uno spazio)

Il *selettore di discendenza* è costituito da *due selettori separati da uno spazio* e *consente di individuare uno o più elementi figli (annidati)* ***di qualunque livello*** *rispetto a un elemento padre.*

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

Leggendo il selettore da destra verso sinistra: si individuano tutti i paragrafi che discendono da un divisore (ovvero sono contenuti da qualche parte all'interno di un divisore).

Child selector (selettori separati da un >)

Il *selettore figlio* è costituito da *due selettori separati da un ">"* e *consente di individuare uno o più elementi figli (annidato)* ***di primo livello*** *rispetto a un elemento padre.*

```
div > p {
  background-color: yellow;
}
```

Leggendo il selettore da destra verso sinistra: si individuano tutti i paragrafi che discendono ***direttamente*** *da un divisore (ovvero sono contenuti direttamente all'interno di un divisore).*

Adjacent sibling selector (selettori separati da un +)

Il selettore di fratello adiacente è costituito da *due selettori separati da un "+"* e *consente di individuare il primo elemento fratello (facente parte* ***dello stesso livello)*** *rispetto a un altro elemento.*

```
div + p {
  background-color: yellow;
}
```

Leggendo il selettore da destra verso sinistra: si individua il primo paragrafo che segue ***direttamente*** *un divisore (ovvero il primo paragrafo, appartenente allo stesso livello, che viene dopo un div).*

General sibling selector (selettori separati da una tilde)

Il selettore generale di fratelli è costituito da *due selettori separati da un "~"* e *consente di individuare uno o più elementi fratelli (facente parte* ***dello stesso livello)*** *rispetto a un altro elemento.*

```
div ~ p {
  background-color: yellow;
}
```

Leggendo il selettore da destra verso sinistra: si individua tutti i paragrafi che seguono ***direttamente*** *un divisore (ovvero i paragrafi, appartenenti allo stesso livello, che vengono dopo un div).*

Selettori basati sugli attributi

Un selettore basato sugli attributi permette di individuare uno o più elementi HTML basandosi *sulla presenza o meno di un attributo o sul contenuto dell'attributo stesso*. I seguenti esempi spiegano le funzionalità dei vari selettori:

- ***a[title]***: *tale selettore individua tutti gli hyperlink in cui è stato* ***definito*** *l'attributo title, il quale può essere anche vuoto;*
- **a[target="VAL"]**: *tale selettore individua tutti gli hyperlink in cui l'attributo* ***target*** ha come valore **VAL***, un'uguaglianza stretta;*
- **a[href^=" VAL"]**: *tale selettore individua tutti gli hyperlink in cui l'attributo* ***href*** inizia con il valore **VAL***;*
- **a[href$=" VAL"]**: *tale selettore individua tutti gli hyperlink in cui l'attributo* ***href*** termina con il valore **VAL***;*
- **a[href*="VAL"]**: *tale selettore individua tutti gli hyperlink in cui l'attributo* ***href*** contiene almeno il valore **VAL***, paragonabile al contains() di Java;*
- **a[href|="VAL"]**: *tale selettore individua tutti gli hyperlink in cui l'attributo* ***href*** equivale o contiene il valore **VAL** *preceduto o meno da un trattino, può essere utile per individuare gli elementi con un attributo custom;*
- **a[class~=" VAL"]**: *tale selettore individua tutti gli hyperlink in cui nell'attributo multi-valore (più valori separati dagli spazi)* ***class*** *vi sia il* **VAL*;***

Prima della chiusura della parentesi quadra è possibile inserire un flag:

- *i: specifica che il matching del selettore è case-insensitive;*
- *s: specifica che il matching del selettore è case-sensitive;*

Selettori di pseudo-classi

Una pseudo-classe è una *keyword* preceduta ":" aggiunta ad un selettore per specificare che le regole di stile deve essere applicate solo quando gli elementi sono in un certo stato. Segue una lista *non completa* delle pseudo-classi esistenti:

- *:active*: *individua il link attivo al momento*;
- *:checked: individua gli input che hanno l'attributo "checked" o che sono stati selezionati;*
- *:disabled: individua gli input disabilitati;*
- *:empty: individua gli elementi HTML che hanno un contenuto vuoto;*
- *:enabled: individua gli input che sono abilitati;*
- *: first-child: individua tutti gli elementi HTML del tipo indicato dal selettore che sono il primo figlio dei rispettivi padri;*
- *: first-of-type: individua tutti gli elementi HTML del tipo indicato dal selettore che sono il primo elemento del tipo indicato dal selettore dei rispettivi padri;*
- *:focus: individua l'elemento che ha attualmente il focus dell'utente;*
- *:hover: individua l'elemento su sui il puntatore sta sopra;*
- *:last-child: individua tutti gli elementi HTML del tipo indicato dal selettore che sono l'ultimo figlio dei rispettivi padri;*
- *:last-of-type: individua tutti gli elementi HTML del tipo indicato dal selettore che sono l'ultimo elemento del tipo indicato dal selettore dei rispettivi padri;*
- *:not(): effettua la negazione della pseudo-classe che contiene fra parentesi;*
- *:nth-child(): individua tutti gli elementi HTML del tipo indicato dal selettore che sono l'n-esimo figlio dei rispettivi padri;*
- *:nth-last-child(): (): individua tutti gli elementi HTML del tipo indicato dal selettore che sono l'n-esimo figlio dei rispettivi padri, contando dall'ultimo;*
- *:nth-last-of-type(): individua tutti gli elementi HTML del tipo indicato dal selettore che sono l'n-esimo elemento del tipo indicato dal selettore dei rispettivi padri, contando dall'ultimo;*
- *:read-only: individua gli input che hanno l'attributo "readonly";*
- *:required: individua gli input che hanno l'attributo "required";*
- *:valid: individua gli input il cui contenuto risulta valido, secondo il pattern fornito;*
- *:visited: individua i link già visitati, ovvero vi è un riscontro nella cronologia del browser;*

Selettori di pseudo-elementi

Uno pseudo-elemento è una *keyword* "::" aggiunta ad un selettore e individua specifiche *parti* di un elemento HTML. Seguono alcuni esempi:

- *::before: indica la parte subito dopo un elemento HTML;*
- *::after: indica la parte che precede un elemento HTML;*
- *::first-letter: indica la prima lettera di un elemento contenente testo;*
- *::first-line: indica la prima linea di un elemento contenente testo;*

Priorità delle regole CSS

Una regola CSS può essere definita in due modalità: inline, quando è definita attraverso l'attributo *style* dell'elemento HTML oppure nel *foglio di stile CSS* (che può essere interno o esterno al documento HTML).

Un elemento HTML può quindi avere regole simili definite in entrambe le modalità, è necessario quindi definire una metrica di priorità che si sleghi dal concetto di applicazione a cascata.

Nel CSS è inoltre definita la keyword ***!important*** che se aggiunta a una regola CSS fornisce la massima priorità indipendentemente dalla modalità in cui è stata definita. La priorità delle regole CSS è così definita:

$$!important > inline > foglio\ di\ stile$$

Importare un foglio CSS esterno

Un foglio di stile CSS può essere:

- *Embeded*: quando all'interno della sezione *head* del documento HTML vi è il tag *<style>,* in cui è possibile scrivere le regole CSS;
- *External*: quando vi è un file CSS separato in cui sono scritte le regole CSS della pagina web, il caricamento di tale file dovrà essere dichiarato nel documento HTML;

Il caricamento di un *foglio di stile esterno* avviene mediante *un tag link* nella sezione *head* del documento HTML. Un template di tale tag è il seguente:

```
<link
  rel="stylesheet"
  [type="txt/css"]
  href="styles.css"
  [media="all"]
  />
```

- *rel: specifica che è un foglio di stile;*
- *type: è opzionale ed indica che è un testo di un foglio di stile;*
- *href: specifica l'indirizzo url del foglio di stile;*
- *media: è opzionale e indica per quando le regole devono essere applicate;*
  - *print: le regole vanno applicate per la stampa del documento;*
  - *screen: le regole vanno applicate per la visualizzazione su schermo;*
  - *projection: le regole vanno applicate per la visualizzazione su proiettore;*

Si può importare un file CSS anche all'interno di un altro file CSS attraverso la direttiva:

$$@import\ 'stile2.css'$$

Unità di misura

Nel linguaggio CSS distinguiamo le unità di misura *assolute* da quelle *relative.*

Unità di misura assolute

Le *unità di misura assolute* sono caratterizzate da valori fissi.

Unità	Descrizione	Unità	Descrizione
cm	centimetri	px	pixel
mm	millimetri	pt	punto
in	pollici	Pc	pica

L'unità pixel è legata al dispositivo di visualizzazione: per i dispositivi low-dpi 1 px equivale a un punto del display mentre per gli schermi ad alta risoluzione 1 pixel equivale a svariati pixel fisici.

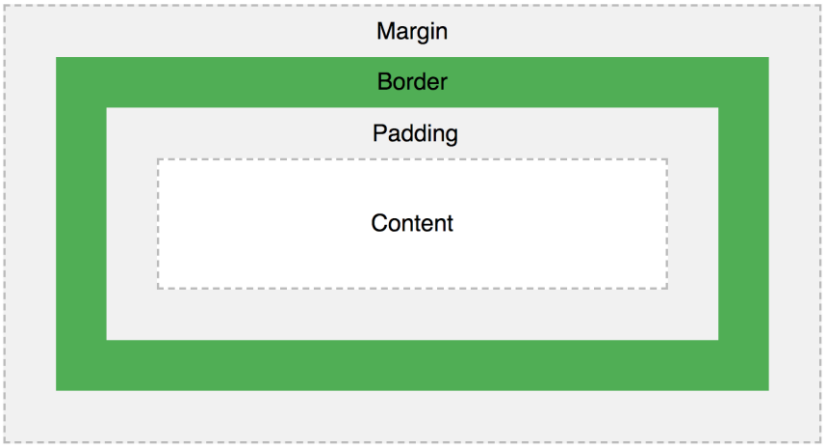
Unità di misura relative

Le *unità di misura relative* sono caratterizzate da valori che dipendono da altri fattori. Per alcune metriche si introduce il concetto di viewport, ovvero la dimensione della finestra del browser.

Unità	Descrizione	Unità	Descrizione
em	Relativa al font-size dell'elemento	vw	Relativa al 1% della lunghezza della viewport
rem	Relativa al font-size dell'elemento root (html)	vh	Relativa al 1% dell'altezza della viewport

Box Model

L'engine dei browser renderizza ogni elemento HTML come un rettangolo seguendo lo standard denominato *box model*. Ogni *box (rettangolo)* è costituito da quattro parti (o aree): *il margine, il bordo, il padding e il contenuto.*



CSS: Margin

Il margine è definito come lo spazio che separa un elemento HTML da un altro nella pagina web.

La regola CSS *margin* è una regola aggregatore può applicare contemporaneamente: *margin-top, margin-right, margin-bottom e margin-left.*

Tale regola può avere un numero variabile di argomenti, il quale comporta una funzionalità differente:

- *margin: 1px, applica a tutte e 4 le direzioni la metrica scelta;*
- *margin: 5% auto, applica la prima metrica al top e bottom e la seconda al right e left;*
- *margin: 1em auto 2em, applica la prima metrica al top, la seconda al right e left e l'ultima al bottom;*
- *margin: 2px 1em 0 auto, applica gli argomenti in senso orario partendo dal top;*

*Oltre a valori metrici è possibile assegnare **auto**, ciò tenterà di centrare l'elemento rispetto al padre.*

CSS: Border

Il border è quello spazio che si trova fra il margine e il padding di un elemento HTML.

La regola CSS *border* è una regola aggregatore può applicare contemporaneamente: *border-width, border-style e border-color.*

Tale regola ha 3 argomenti che sono i rispettivi argomenti delle regole CSS che aggrega:

- *<border-width>: specifica lo spessore del bordo;*
- *<border-style>: specifica lo stile del bordo, il quale può essere solid, dotted, dashed ect…*
- *<border-color>: specifica il colore del bordo;*

CSS: Padding

Il padding è definito come lo spazio che si trova fra il bordo e il contenuto di un elemento HTML.

La regola CSS *padding* è una regola aggregatore può applicare contemporaneamente: *padding-top, padding -right, padding -bottom e padding -left.*

Tale regola può avere un numero variabile di argomenti, il quale comporta una funzionalità differente:

- *padding: 1px, applica a tutte e 4 le direzioni la metrica scelta;*
- *padding: 5% 10%, applica la prima metrica al top e bottom e la seconda al right e left;*
- *padding: 1em 0 2em, applica la prima metrica al top, la seconda al right e left e l'ultima al bottom;*
- *padding: 2px 1em 0 5px, applica gli argomenti in senso orario partendo dal top;*

Colori nel CSS

Nel linguaggio CSS è possibile definire un colore in vari modi, attraverso un nome o con l'utilizzo degli spazi colore:

- Text: tomato
- RGB: rgb(255, 99, 71)
- HEX: #ff6347
- HSL: hsl(9, 100%, 64%)
- RGBA: rgba(255, 99, 71, .5) //trasparent
- HSLA: hsla(9, 100%, 64%, .5) //trasparent

Regole CSS applicabili ai testi

- *color: definisce il colore del testo;*
- *text-decoration: specifica una decorazione come l'underline, wavy o nessuno;*
- *text-transform: specifica una trasformazione come l'uppercase o il capitalize;*
- *text-shadow: specifica l'ombreggiatura del testo;*
- *text-align: specifica l'allineamento del testo;*
- *line-height: specifica l'altezza di una riga;*
- *letter-spacing: specifica lo spazio tra i caratteri;*
- *word-spacing: specifica lo spazio tra le parole;*
- *font-size: specifica la dimensione del testo;*
- *font-weight: specifica il peso del testo, ovvero l'intensità del grassetto;*
- *font-family: specifica il font del testo;*

CSS: Display

La regola *display* specifica il *tipo di rendering* da applicare ad un elemento HTML (i.e. block o inline). In HTML, la proprietà display assume di default i valori specificati dagli standard o da un foglio di stile dell'utente.

Tale regola ha numerosi valori che determinano comportamenti differenti, vediamo quelli più usati:

Valore	Descrizione
block	Visualizza l'elemento HTML come se fosse di tipo *block*
inline	Visualizza l'elemento HTML come se fosse di tipo inline
none	Rimuove l'elemento dalla pagina web, rimuovendo tutto ciò che lo riguarda
inline-block	Visualizza l'elemento HTML come se fosse inline permettendo di modificarne altezza e lunghezza
flex	Visualizza l'elemento come un contenitore flexbox
grid	Visualizza l'elemento come un contenitore gridbox

CSS: Visibility vs Opacity

Le regole *visibility e opacity* agiscono sulla visibilità di un elemento HTML. La prima regola ad oggi risulta essere essenzialmente deprecata.

La regola *visibility* può avere tre valori auto-esplicativi: ***hidden, visible e collapsed***. Nascondere un elemento HTML con tale regola inibisce il funzionamento degli eventi sull'elemento.

La regola *opacity* specifica il livello di opacità di un elemento e si applica in maniera ricorsiva anche a tutti i figli dell'elemento. Impostando il valore a 0 l'elemento risulterà invisibile ma, al contrario della regola *invisibility,* non collassa (mantiene le sue proporzioni) e mantiene gli eventi su di esso.

CSS: Position

La regola di posizionamento definisce come un elemento è posizionato nel documento HTML. Di default, un elemento HTML ha come valore *di position* il valore *static.* Vi sono diversi metodi di posizionamento: *relativo, assoluto, fissato e sticky*.
La regola di posizionamento fa uso di alcune *sotto regole* per definire lo spostamento rispetto a una determinata posizione:

- *top*: spostamento rispetto all'alto;
- *bottom*: spostamento rispetto il basso;
- *left*: spostamento rispetto alla sinistra;
- *right*: spostamento rispetto alla destra;

Posizionamento relativo

In un *elemento HTML con posizionamento relativo* le *sotto regole* specificano un'offset rispetto alla posizione naturale dell'elemento nella pagina web.

Posizionamento assoluto vs Posizionamento fissato

In un *elemento HTML con posizionamento assoluto* le *sotto regole* specificano un'offset rispetto al *primo elemento antenato* (nell'albero html) con *posizionamento non static.* La ricerca di un antenato può risalire l'albero fino a raggiungere la viewport.

Un elemento con posizionamento assoluto è rimosso dal flusso di rendering della pagina e viene renderizzato a parte, le sue dimensioni non influenzano il layout della pagina web.

Un *elemento con posizionamento fissato* si comporta esattamente come un *elemento con posizionamento assoluto* con l'unica differenza che le *sotto regole* specificano *sempre* un offset rispetto alla viewport.

In un *elemento con posizionamento assoluto o fissato:*

- *s*e l'altezza è definita la *regola top prevale sulla regola bottom;*
- se la larghezza è definita, *la regola a prevalere dipende dal verso di scrittura della lingua in uso;*

Riempimento container in posizionamento assoluto o fissato

Per ottenere un *effetto full-screen o il riempimento del container* da parte di un *elemento con posizionamento assoluto o fissato* è possibile porre le 4 *sotto regole a 0*.

```
div {position: absolute;top:0;bottom:0;left:0;right:0;background-color:red;}
```

Posizionamento Sticky

In un *elemento con posizionamento sticky* la posizione dell'elemento dipende dallo *scrolling* dell'utente. Il comportamento *sticky* è uno switch fra un *posizionamento relativo e un posizionamento fissato legato allo scrolling: l'elemento è posizionato in modo relativo fino a quando un offset rispetto alla viewport non viene triggerato, successivamente l'elemento assume un posizionamento fissato rispetto alla viewport "appiccicandosi" ad esso.*

CSS: Float e clear

La regola *float* specifica che un elemento debba "fluttuare" nella parte destra o sinistra del suo contenitore, permettendo agli elementi inline di posizionarsi attorno ad esso.

La regola può avere svariati valori ma i più comuni sono: left, right, none, quest'ultima è quella predefinita.

L'elemento fluttuante è rimosso dal normale flusso di rendering della pagina, ma le sue dimensioni influenzano comunque il layout della pagina web.

La regola *clear* specifica se un elemento può stare accanto agli *elementi fluttuanti* che lo precedono o deve essere spostato verso il basso (*cleared*) sotto di essi. *La proprietà clear si applica sia agli elementi floated che a quelli non floated.*

CSS tricks: clearfix

Se un elemento contiene solo elementi floated, la sua altezza collasserà a 0. Se si vuole preservare la dimensione, così da avere gli elementi floated all'interno di esso, allora è necessario effettuare un self-clear dei nodi figli. Tale trick è chiamato *clearfix*. Per applicare il clearfix basta applicare le regole content, clear e display al pseudo-elemento *after del container.*

```
.container::after {
  content: "";
  clear: both;
  display: block;
}
```

CSS: Background

Le regola di *background* permettono di specificare tutte le caratteristiche legate al background di un elemento.

Il background di un elemento può essere:

- background-color: *specifica che il background è un colore;*
- *background: url(…): specifica che il background è un'immagine;*

Qualora il background sia un'immagine il CSS prevede ulteriori regole:

- *background-size*: specifica le dimensioni dell'immagine, può avere come parametri le due dimensioni fisse, relative o oppure keyword come *contain, cover*;
- *background-position*: specifica la posizione iniziale dell'immagine;
- *background*-repeat: specifica se l'immagina possa ripetersi o meno per riempiere eventuale spazio non coperto;

CSS: Cursor

La regola *cursor permette di specificare quale cursore del puntatore visualizzare quando si ci trova sopra un elemento HTML.*

```
#textContainer {cursor: text;}
```

CSS: Overflow

La regola *overflow* specifica il *comportamento dell'elemento qualora il suo contenuto sia troppo grande per lo spazio concesso all'elemento*. È possibile anche utilizzare *overflow-x e overflow-y* per specificare il comportamento in base all'asse in cui avviene l'overflow.

```
div {background-color: #eee;width: 200px;height: 50px;border: 1px dotted black;overflow: visible;}
```

Tale regola può assumere vari valori, ma i più diffusi sono:

- visible*: il contenuto che fuoriesce è visibile;*
- *hidden: il contenuto che fuoriesce è nascosto;*
- *clip: simile all'hidden ma vieta lo scrolling;*
- *scroll: viene introdotta la funzionalità di scrolling;*

CSS: z-index

La regola *z-index* specifica l'ordine nell'asse z, quindi l'ordine di visualizzazione per gli elementi sovrapposti. Tale regola accetta valori numeri e il valore *-1* rappresenta il livello con minore priorità.

```
img {position: absolute;left: 0px;top: 0px;z-index: -1;}
```

CSS Responsive

Con il passare del tempo e l'avanzamento tecnologie sempre più dispositivi ottengono la capacità di connettersi al web. Dispositivi come gli smartphone, pc con schermi ad alta risoluzione, smartwatch e TV Smart hanno man mano evidenziato un'importante problematica: in passato i siti web erano sviluppati basandosi su una risoluzione e si consigliava la navigazione con quest'ultima, ma ora i dispositivi sono molteplici con altrettante risoluzioni.

È quindi sorta l'esigenza di dimensionare parti del sito web a seconda del dispositivo che lo sta visualizzando.

Con il termine *responsive* di indica un sito capace di *adattarsi* alla risoluzione del dispositivo che visualizza il sito al fino di fornire la migliore accessibilità possibile.

Nel corso vedremo i 3 principali layout responsive.

Responsive Layout: Columns System

Il columns system è il layout responsive molto vecchio ma assai affidabile e supportato pressoché da qualunque browser. L'idea alla base è quella di organizzare le componenti della pagina su nuove unità di misura, canonicamente 12 *colonne.*

La pagina è suddivida in *righe* aventi lunghezza 100% e gli elementi HTML sono distribuiti nelle *colonne della riga*, le colonne sono *contenitori fluttuanti verso sinistra* con *lunghezza variabile. Una riga è formata da una qualsiasi combinazione di tipi di colonne la cui somma delle lunghezze sia pari al 100%.*

Poiché una riga ha come figli diretti solo *elementi fluttuanti è necessario applicare il **clearfix** affinché l'altezza della riga non collassi (ovvero risulti 0).*

```
.row::after {
    content: "";
    clear: both;
    display: block;
}
```

Dopo aver applicato il clearfix, usiamo *il selettore basato sugli attributi* per individuare tutte gli *elementi* colonna per specificare *il float e un padding.* Infine tramite dei *class selector* definiamo la lunghezza delle di ogni *tipo di colonna*:

```
[class*="col-"] {
    float: left;
    padding: 15px;
}
```

```
.col-1  {width: 8.33%;  }
.col-2  {width: 16.66%; }
.col-3  {width: 25%;    }
.col-4  {width: 33.33%; }
.col-5  {width: 41.66%; }
.col-6  {width: 50%;    }
.col-7  {width: 58.33%; }
.col-8  {width: 66.66%; }
.col-9  {width: 75%;    }
.col-10 {width: 83.33%; }
.col-11 {width: 91.66%; }
.col-12 {width: 100%;   }
```

Media Query

L'elemento che rende *responsive* il *columns system* è l'utilizzo delle *media query*. Attraverso le *media query* si definiscono dei *breakpoint* basati sulla risoluzione dello schermo del dispositivo che forniscono *regole di stile ad hoc per rendere il sito più accessibile possibile per la risoluzione del dispositivo.*

*Seguendo il concetto di **Mobile First**, un sito web dovrebbe essere sviluppato pensando al mobile e successivamente scalata per le risoluzioni più grandi mediante le media query. Per questo motivo le media query devono essere piazzate nella parte finale del foglio di stile e in ordine crescente rispetto al breakpoint.*

Una *media query* viene definita come segue:

```
@media TYPE and (min-width: 900px) {
    ...
}
```

- *type:* specifica quando applicare le regole di stile, può essere *screen, print, all, projector, ect…*
- *(…): è una condizione per cui la media query viene attivata o meno;*

Per il columns system, usiamo le media query basate su breakpoint noti per modificare lo *stile delle colonne e degli elementi HTML per adattarli meglio alla visualizzazione su una determinata risoluzione.*

Responsive Layout: Flexbox System

Il *flexbox system* è un layout responsive recente adatto per gestire i *componenti di una applicazione* o per i *layout* di piccola scala che devono essere disposti in un ambiente unidimensionale, o in riga o in colonna.

La convenienza del flexbox è che permette in maniera assai facile senza grandi complessità le seguenti operazioni:

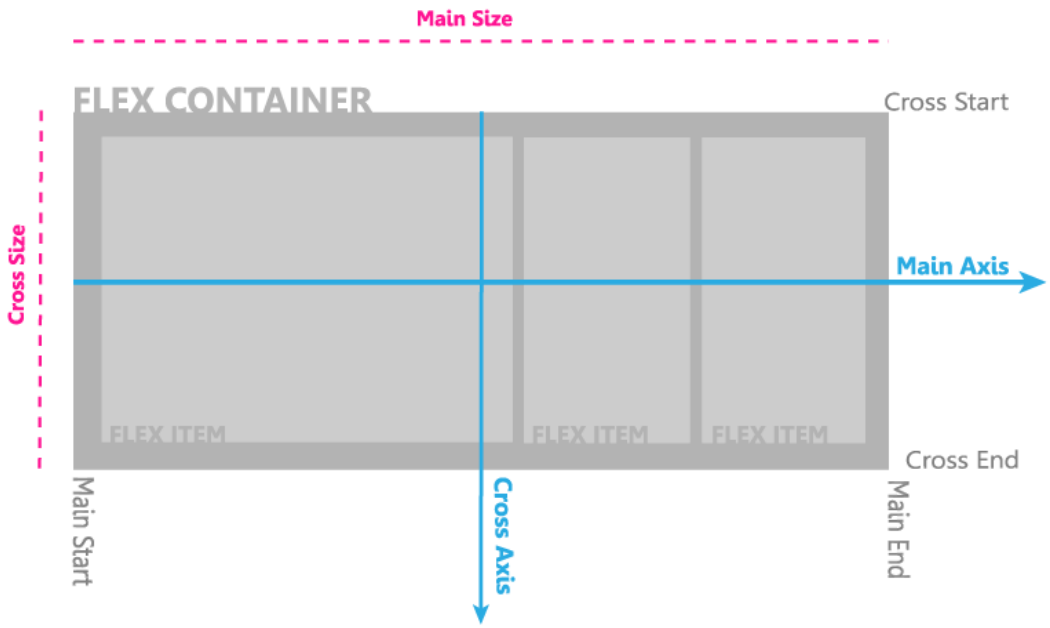
- Centrare verticalmente un blocco di contenuti dentro l'elemento padre;
- Far prendere a tutti i figli di un contenitore equal altezza/lunghezza indipendentemente dall'altezza e lunghezza totale disponibile;
- Fare adottare a tutte le colonne di un sistema a colonne la stessa altezza anche se quest'ultime contengono una quantità di elementi differenti;

Flexbox model

Il *flexbox model* consiste in un *flex-container che contiene dei flex-item.* Per creare un *flex-container* si specifica la *regola display*.

Tutti i figli diretti del *flex-container* saranno automaticamente considerati come *flex-items.*

```
.flex-container {
  display: flex;
}
```



- Si definisce *asse principale* l'asse che va per la direzione con la quale vengono visualizzati gli oggetti, *normalmente da sinistra verso destra;*
- Si definisce *asse trasversale* l'asse che va perpendicolarmente alla direzione con la quale vengono visualizzati gli oggetti, *normalmente dall'alto verso il basso;*

Regole del flex-container

Le regole del *flex-container* permettono di gestire la disposizione dei *flex-item. Vi sono svariate regole con funzionalità diversa:*

- *flex-flow: è una regola aggregatore che accetta fino a 2 parametri rispettivamente delle regole flex-direction e flex-wrap;*
    - *flex-direction:* specifica *la direzione* in cui vogliamo *impilare* i *flex-item*. Può assumere: ***row, row-reverse, column e column-reverse***. La direzione colonna inverte le assi. I valori *reverse* invertono l'ordinamento dei *flex-item*;

    - *flex*-wrap: specifica il *comportamento in caso di overflow* dei *flex-item* rispetto alle dimensioni del *flex-container.* Può assumere: ***nowrap, wrap e wrap-reverse****.* Il wrap permette ai *flex-item in overflow di andare a capo;*
- *justify-content:* specifica la disposizione dei *flex-item* per l'*asse principale*. Può assumere i seguenti valori:
    - ***flex-start:*** gli item *sono posizionati a partire dall'inizio* del container;
    - ***flex-end:*** gli item *sono posizionati a partire dalla fine* del container;
    - ***center:*** gli item *sono posizionati al centro* del container;
    - ***space-between:*** gli item *presentano uno spazio fra di loro;*
    - ***space-around:*** gli item *presentano uno spazio prima, fra e dopo di loro;*
    - ***space-evenly:*** gli item presentano un *equal spazio attorno a loro;*
- *align-items:* specifica la disposizione dei *flex-item* per l'*asse trasversale*. Può assumere i seguenti valori:
    - ***flex-start:*** gli item *sono posizionati a partire dall'inizio* del container;
    - ***flex-end:*** gli item *sono posizionati a partire dalla fine* del container;
    - ***center:*** gli item *sono posizionati al centro* del container;
    - ***stretch:*** gli item sono stretchati fino a riempire il *flex-container;*
- *align-content:* specifica *la disposizione delle flex-linee rispetto l'asse trasversale. Assume gli stessi valori di align-items;*

Regole del flex-item

Le regole del *flex-item* permettono di gestire il loro comportamento*. Vi sono svariate regole con funzionalità diversa:*

- *flex: è una regola aggregatore che accetta fino a 3 parametri rispettivamente delle regole flex-grow, flex-shrink e flex-basis;*
    - *flex-grow:* specifica *quanto un item debba crescere rispetto agli altri item dello stesso flex-container.* Assume valore numerici che rappresentano la proporzione:
    - *flex*-shrink: specifica *quanto un item debba rimpicciolirsi rispetto agli altri item dello stesso flex-container.* Assume valore numerici che rappresentano la proporzione:
    - *flex-basis:* specifica la *lunghezza iniziale* di un *item;*
- *align-self:* specifica la disposizione del *flex-item* sovrascrivendo la proprietà *align-items* del *flex-container*. Possono assumere i seguenti valori:
    - ***flex-start:*** l'item *è posizionato all'inizio* del container;
    - ***flex-end:*** l'item *è posizionato alla fine* del container;
    - ***center:*** l'item *è posizionato al centro* del container;
    - ***stretch:*** l'item è stretchato fino a riempire il *flex-container;*
- *order:* specifica la posizione dell'item rispetto agli altri item del *flex-container;*

Responsive Layout: Grid System

Il *grid system* è un layout responsive più recente adatto per gestire *layout* di grande scala che devono essere disposti in un ambiente bidimensionale.

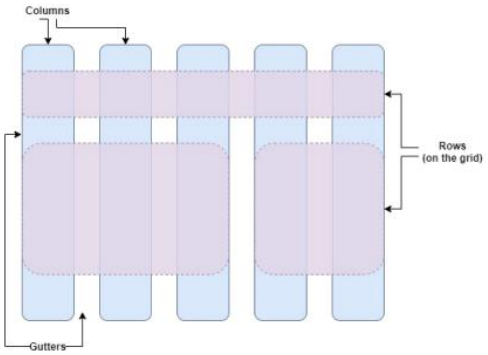
Essendo il più recente ancora non vi è una vera e propria standardizzazione, perciò molti browser lo hanno implementato secondo i propri gusti. Ciò comporta una lenta adozione da parte degli sviluppatori in quanto non è garantita la compatibilità fra i vari web browser.

Grid Model

Una griglia è tipicamente costituita da un *grid-container* che contiene le righe e le colonne, sotto forma di linee, e le spaziature denominate *gap o gutters*. Per creare un *grid-container* si specifica la *regola display*.

```
.container {
    display: grid;
}
```

Tutti i figli diretti del *grid-container* saranno automaticamente considerati come *grid-items.*



Regole del grid-container

Le regole del *grid-container* permettono di gestire la forma della griglia*:*

- *grid-template-[columns|rows]:* specifica *il numero di [columns|rows] della griglia*. Accetta come parametro una lista di lunghezze separate da spazi, che rappresentano le singole lunghezze di ogni *[columns|rows]*;
    - Per poter definire un certo numero di *[columns|rows]* con la stessa lunghezza è possibile usare la funzione CSS: ***repeat(quantità, lunghezza);***
    - Per poter riempire una griglia di quante più *[columns|rows] possibile, si può usare la seguente funzione CSS:* ***repeat(auto-fill, minmax(lunghezza_minima, lughezza_massima))***
- *gap*: specifica la dimensione della spaziatura fra righe e della spaziatura fra colonne. Accetta uno o due parametri: con un solo parametro applica la dimensione ad entrambe le spaziature, con due parametri applica il primo alla spaziatura di riga e il secondo alla spaziatura di colonna;
- *grid*-template-areas: specifica una descrizione a pattern della griglia, ci permette di assegnare etichette e span alle celle della griglia.
- *justify-items e align-items:* specificano la disposizione dei *grid-item rispettivamente* nella l'asse delle righe e nell'asse delle colonne. Può assumere vari valori:
    - ***start:*** *gli item sono posizionati a filo con il bordo iniziale della loro cella;*
    - ***end:*** *gli item sono posizionati a filo con il bordo finale della loro cella;*
    - ***center:*** *gli item sono posizionati al centro della loro cella;*
    - ***stretch:*** *gli item riempiono la loro cella;*

Regole del grid-item

Le regole del *grid-item* permettono di gestire il posizionamento dell'item nelle celle*:*

- *grid-[column|row]:* specifica il range di *[column|row]* che l'item ricopre. Accetta due parametri separati da un forward slash: il primo indica la *[column|row]* di inizio e il secondo quella di fine, l'item ricoprire le *[column|row]*che vanno dalla *[column|row]* di inizio fino a quella prima della fine;
    - *Inserendo la keyword* ***span*** *dopo il forward slash, il secondo parametro cambia funzionalità: specifica di quante [column|row] dovrà estensersi l'item;*
- grid-area: specifica il posizionamento dell'item in un'area definita nel *grid-template-areas del gird-container*. Accetta come parametro una stringa che equivale all'etichetta dell'area;
- *justify-self e align-self:* specificano la disposizione dell'item *rispettivamente* nella l'asse delle righe e nell'asse delle colonne, sovrascrivendo le controparti definite nel *grid-container* . Può assumere vari valori:
    - ***start:*** *l'item è posizionato a filo con il bordo iniziale della sua cella;*
    - ***end:*** *l'item è posizionato a filo con il bordo finale della sua cella;*
    - ***center:*** *l'item è al centro della sua cella;*
    - ***stretch:*** *l'item riempie la sua cella;*

JavaScript

Introduzione

JavaScript (JS) è un *linguaggio di programmazione interpretato* di comune uso nelle pagine web. JS è un linguaggio *multi-paradigma (*programmazione funzionale, imperativa, asincronicità), *dinamicamente tipizzato* (o *debolmente tipizzato),* ovvero il *tipo delle variabili* è valutato a *runtime*, e a *singolo thread* di esecuzione*, ovvero può eseguire solo un'istruzione alla volta, non bloccante.*

Nei browser vi è un *engine JavaScript* che si occupa di eseguire il codice JS dopo il caricamento del codice HTML e del foglio di stile, ciò assicura che tutto sia in posizione per il corretto funzionamento del codice JS. Infatti, se JS fosse eseguito prima del caricamento dell'HTML e del CSS potrebbero accadere degli errori di runtime.

L'esecuzione del codice JS di una pagina web è relativamente sicuro poiché il browser esegue ogni finestra dentro una *sandbox*, che non permette l'accesso diretto al sistema host dell'utente.

Fanno eccezione le *estensioni del browser* (*anch'esse scritte in JS)* che hanno accesso a un *set di API più ampio* che *possono essere usate in modo malevolo.*

JavaScript nelle pagine web

Un codice JS può essere aggiunto in una maniera simile a quella del CSS: codice embeded o esterno:

- *embeded*: è possibile usare il tag $< script > \cdots </script >$ il cui contenuto è codice JS;
- *esterno:* caricare un file js separato atraverso l'attributo *src* del tag $< script >$

Internal:

```
<script>
  // JavaScript goes here
</script>
```

External:

```
<script src="script.js"></script>
```

JS può essere definito nella sezione *head* del documento HTML, *ma è consigliabile metterlo prima del tag di chiusura del body per favorire la velocità di visualizzazione della pagina: è meglio dare la precedenza al caricamento degli elementi HTML che ai file js.*

Commenti nel codice JS

JS prevede *commenti a singola riga e commenti multi riga.* Non si possono annidare commenti.

```
// a one line comment

                        /* You can't, however, /* nest comments */ SyntaxError */
/* this is a longer,
   multi-line comment
```

Variabili nel codice JS

In JS le variabili *devono iniziare con una lettera, un underscore (_) o un segno del dollaro ($).*

Dichiarazione: var vs let vs const

In JS è possibile dichiarare una variabile in 3 modi:

- *var:* può definire *variabili globali o locali, quest'ultime aventi un function-scope*;
- *let:* può definire *variabili locali* aventi un *block-scope*, ovvero sopravvivono fino alla chiusura del blocco più vicino;
- *const:* può definire *costanti locali* aventi un *block-scope*, ovvero sopravvivono fino alla chiusura del blocco più vicino;

Le *variabili dichiarate ma non inizializzate* hanno come *valore predefinito* il valore *undefined*.

JS Hoisting

L'*hoisting* di JS è un meccanismo che consiste nello *spostare "logicamente"* le *dichiarazioni di variabili (dichiarate con var) e funzioni* nella parte superiore del codice al fine di poter usare accedere alle variabili e funzioni in qualunque punto del codice, anche se la dichiarazione è collocata dopo successivamente all'accesso.

```
var a;
console.log('The value of a is ' + a); // The value of a is undefined

console.log('The value of b is ' + b); // The value of b is undefined
var b; Hoisting :)

console.log('The value of c is ' + c); // Uncaught ReferenceError: c is not defined

let x;
console.log('The value of x is ' + x); // The value of x is undefined

console.log('The value of y is ' + y); // Uncaught ReferenceError: y is not defined
let y; No hoisting :(
```

Tipi primitivi e conversione

JS prevede come tipi di dati primitivi:

- *string:* una stringa che è consiste in una lista (con relativi metodi e accessi) ordinata di caratteri;
- *number:* un numero che può essere intero o float;
- *boolean:* un booleano vero o falso;
- *undefined;*
- null;
- object;
- function;

Essendo JS un linguaggio *dinamicamente tipizzato*, la conversione del tipo di una variabile è eseguita automaticamente durante l'esecuzione del codice.

If, switch, while, for

I costrutti e il funzionamento degli *if condizionale, switch, ciclo while e ciclo for* sono paragonabili a quelli degli altri linguaggi. Si è preferito omettere tale parte.

Array JS

Per la creazione degli *array*, JS fa uso del *JavaScript Array object* che crea *oggetti basati su liste di alto livello*.

La dichiarazione di un array avviene mediante le parentesi quadre: **[…]**.
Gli array JS sono infatti:

- oggetti, *typeof* ritorna "object";
- iterabili, ovvero navigabili tramite cicli convenzionali o forEach;
- accessibili tramite indici;
- operabili tramite funzioni come quelle di uno stack o altro;

```
var fruits = ["Apple", "Banana"];
console.log(fruits.length);
// 2

var first = fruits[0];
// Apple

var last = fruits[fruits.length - 1];
// Banana

fruits.forEach(function (item, index, array) {
  console.log(item, index);
});
// Apple 0
// Banana 1
```

Funzioni JS

La dichiarazione di una funziona JS consiste nella keyword *function* seguita da:

- il *nome* della funzione;
- una *lista di parametri* separati da virgole racchiusa in parentesi tonde;
- il *corpo della funzione* racchiuso dentro parentesi graffe;

```
function square(number) {
  return number * number;
}
```

Come detto in precedenza, le dichiarazioni delle funzioni sono soggette all'hoisting.

Funzioni anonime

Le *funzioni anonime* sono funzioni definite attraverso un'*espressione di funzione*, la quale può essere salvata all'interno di una variabile. Questo permette di passare *funzioni* come *argomenti di altre funzioni*, un esempio può essere l'utilizzo di *callbacks*.

```
var square = function(number) { return number * number; };
var x = square(4); // x gets the value 16
```

È possibile fornire o meno il nome della funzione, il nome può essere utilizzato per eseguire la *ricorsività* della funzione.

Arrow function

Una *arrow function* è un'espressione più compatta rispetto alla normale *espressione di funzione*. Sono poco adatte ad essere usate come *metodi* di oggetti e non possono essere usate come *costruttore*. Una caratteristica che le differenzia rispetto a una *normale funzione* è il comportamento del *binding del this.*

La sintassi di una arrow function è abbastanza basilare: vi è la lista di parametri separati da virgole racchiusa dentro parentesi tonde, seguita da una *freccia* e con il corpo della funzione racchiuso dentro parentesi graffe.

- Se la funzione *ha solo un parametro*, le *parentesi tonde* possono essere omesse*;*
- Se la funzione *non ha parametri*, *si mantengono le parentesi tonde vuote*;
- Se il corpo della funzione *ha solo un'istruzione che ritorna qualcosa*, le *parentesi graffe e la keyword return possono essere omessi;*

```
(param1, param2, ..., paramN) => { statements }
(param1, param2, ..., paramN) => expression
```

```
singleParam => { statements }
() => { statements }
```

THIS: normal function vs arrow function

Nelle *normali funzioni JS*, la keyword *this rappresenta l'oggetto che ha chiamato la funzione*: che può essere un *elemento DOM* o l'istanza di una classe.

Nelle *arrow function* la keyword *this rappresenta sempre l'oggetto che l'ha definita.*

Closure JS

Le *closure* rappresentano una delle feature più potente di JS, consistono nell'annidare le funzioni. Una *funzione annidata* ha sia l'accesso *alle variabili e ulteriori funzioni annidate* definite all'interno della *funzione padre sia l'accesso alle variabili e alle funzioni a cui ha accesso la funzione padre*.

```
var pet = function(name) {   // The outer function defines a variable called "name"
  var getName = function() {
    return name;             // The inner function has access to the "name" variable of the outer function
  }
  return getName;            // Return the inner function, thereby exposing it to outer scopes
}
myPet = pet('Vivie');

myPet();                     // Returns "Vivie"
```

Oggetti JS

In JS un'oggetto non è altro che una *collezione* di coppie *chiave-valore*, in cui il valore può essere *un qualunque tipo di dato primitivo*, pure una funzione!

La sintassi di un oggetto è alquanto semplice: una lista di coppie *chiave-valore* separati da *virgole* racchiusa dentro *parentesi graffe.*

```
var obj= { [key]:[value][,...]};
```

JavaScript si basa sull'ereditarietà prototipale, ovvero essenzialmente in JS *tutto è un Object* e ogni oggetto conserva un *link* al suo prototipo, ciò crea una *catena di prototipi* che permette al Object di ereditare comportamenti dagli altri oggetti.

Per accedere alle proprietà di un oggetto possiamo usare sia la *notazione del punto (come in Java)* sia la *notazione delle parentesi,* come se fosse un *array associativo* in cui la *chiave è il nome della proprietà*.

Classi JS

Il *costrutto delle classi* in JS è essenzialmente *zucchero sintattico* per *l'ereditarietà basata sui prototipi*. Questo costrutto non introduce in alcun modo il *paradigma OOP.*

Inoltre le definizioni delle classi non sono soggette all'hoisting, perciò la definizione di una classe deve necessariamente precedere l'utilizzo della stessa.

```
class Student {
        name;
        surname;
        #gender; //private
        constructor(name, surname) {
                this.name = name;
                this.surname = surname;
        }
}
```

DOM: Manipolazione HTML tramite JS

Il *DOM (Document Object Model)* è un'*interfaccia di programmazione* per i *documenti HTML*. Rappresenta la pagina, sotto forma di *nodi di un albero ed oggetti*, al fine di permettere ai programmi di *manipolare la struttura, stile e contenuto del documento HTML*.

L'implementazione del *DOM* differisce da *browser a browser* ma tutti seguono *essenzialmente l'attuale standard del DOM*. Attraverso il DOM, i browser rendono i contenuti della pagina web accessibili e manipolabili con *JavaScript*.

Il *DOM* fornisce la seguente serie di *oggetti:*

- *document*: un oggetto che *rappresenta il documento HTML;*
- *window:* un oggetto che *rappresenta* la finestra del browser;
- *Element*: un oggetto *derivante dall'interfaccia del nodo generico* che fornisce metodi e proprietà;

I principali metodi e proprietà visit a lezione sono:

- *document.getElementById(id)*
- *document.getElementsByTagName(tagName)*
- *document.getElementsByClass(className)*
- *document.createElement(tagName)*
- *document.querySelector(selector)*
- *parentNode.appendChild(node)*
- *element.innerHTML*
- *element.style.CSS_RULE*
- *element.setAttribute(attrName, value)*
- *element.getAttribute(attrName)*
- *element.addEventListener(eventName, callback)*
- *window.content*
- *window.onload*
- *console.log()*

Eventi JS

Modello di concorrenza

Essendo JS un linguaggio a *singolo thread di esecuzione* può eseguire solo un'istruzione alla volta ma abbiamo sicuramente visto o navigato siti in cui JS eseguiva più compiti alla volta… com'è possibile?

JavaScript ha un *modello di concorrenza basato su un event loop*, una feature che permette ai browser di *eseguire task a lunga durata separatamente dal thread principale di JS* e, quando l'esecuzione del task è terminata, di mettere in coda nel *thread principale* l'esecuzione di una funzione di *callback*.

Quando viene eseguito del codice JS vengono usate due regioni di memoria:

- *stack*: una memoria ad alte prestazioni usata per eseguire le funzioni e salvare una copia del frame di funzioni (e una copia delle loro variabili locali);
- *heap*: un *memory pool* non strutturato che contiene oggetti e dati primitivi delle closure. Questa memoria è soggetta al *garbage collector,* vi sono tentativi automatici di liberare spazio periodicamente;

Il runtime JavaScript usa una *coda di messaggi,* una lista di *messaggi che devono essere processati.* Ad ogni messaggio è associata una *funzione* che deve essere richiamata per *gestire il messaggio*.

Il runtime processa i messaggi dal più vecchio al più recente: rimuove il messaggio dalla coda ed esegue la f*unzione associata, creando un nuovo stack frame per la sua esecuzione, passandoci come parametro il messaggio stesso.* L'esecuzione del messaggio continua fin tanto che lo stack non è vuoto, quando lo stack sarà vuoto l'*event loop passerà all'esecuzione del prossimo messaggio.*

JavaScript viene definito come un linguaggio non bloccante poiché le operazioni I/O o di altro genere sono essenzialmente gestite tramite eventi che permettono a JS di processare altre cose nell'attesa che l'operazione sia completata.

Registrazione di un Event nel DOM

Per registrare un event handler per un *elemento del DOM* ci sono 3 metodi:

- **addEventListener():** *una funzione che imposta un handler da eseguire qualora l'evento indicato avvenga sul target;*

```
myButton.addEventListener('click', greet);
function greet(event){
      // print and have a look at the event object
      // always print arguments in case of overlooking any other arguments
      console.log('greet:', arguments);
      alert('hello world');
}
```

  - Tale metodo presenta una problematica: rieseguendo la funzione non si andrà a sovrascrivere l'handler definito in precedenza, bensì se ne andrà ad aggiungere uno nuovo alla coda di esecuzione;
- **inline attribute:** *usando gli attributi inline nei tag degli elementi HTML*. Hanno una struttura *on**event=”…”***.

```
<button onclick="alert('Hello world!')">
```

  - Questo metodo è da evitare in quando rende confusionario e poco leggibile il codice markup;
- **Element event function:** *definendo l'handler nella proprietà dell'evento desiderato dell'elemento del DOM.*

```
myButton.onclick = function(event){alert('Hello world');};
```

  - Questo metodo permette *solo un handler per elemento e per evento;*

Asynchronous JS

Ad oggi JavaScript presenta 3 stili per il codice asincrono:

- *Callback (old-style)*
- *Promise;*
- *Async/Await*

CallBack

Lo stile del callback si basa su una funzione, denominata *callback*, che viene passata come *parametro* a una *funzione che verrà eseguita in background (evento o task a lunga durata),* la quale, al suo termine, eseguirà il callback per notificare il completamento dell'operazione.

Quando passiamo un callback come parametro, *passiamo soltanto il riferimento alla funzione,* quest'ultima non viene eseguita immediatamente.

Un esempio di callback è stato già incontrato: *addEventListener() ha come secondo parametro un callback!*

Promise

Lo stile delle promise si basa sull'oggetto P*romise che* fa da tramite per un *valore* che non si conosce al momento della creazione della promise. Inoltre la promise ci permette di associare *handler asincroni* per il successo o meno dell'operazione.

Questo stile comporta che un *metodo asincrono ritorni un valore* come i *metodi sincroni*, il valore ritornato è *una promise.* Una promise può trovarsi in uno dei seguenti stati:

- *Pending*: lo stato iniziando, quando non è né completata né rifiutata;
- *Fulfilled*: quando l'operazione è stata completata con successo;
- *Rejected*: quando l'operazione è fallita per qualche motivo;

La creazione di una promise avviene passando come *parametro* al costruttore della classe Promise una funziona che:

- Come argomenti ha due funzioni callback denominate: *resolve e reject:*
  - *resolve*: è la funzione che *specifica lo stato di fulfilled* e può avere come parametro il risultato dell'elaborazione;
  - *reject*: è la funzione che *specifica lo stato di rejected* e può avere come parametro l'errore incontrato;

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('foo');
  }, 300);
});
```

- *Come corpo ha la logica da eseguire in background che ha il compito di invocare resolve o reject* a seconda del risultato ottenuto;

Una promise dispone sei seguenti metodi:

- *then*: assegna i callback per gli stati di *fulfilled e rejected*. Ritorna una promise;
- *catch*: assegna un callback per lo stato di *rejected*. Ritorna una promise;
- *finally*: assegna il callback da eseguire alla fine della promise indipendentemente dallo stato;

```
myPromise
  .then(handleResolvedA, handleRejectedA)
  .then(handleResolvedB, handleRejectedB)
  .then(handleResolvedC, handleRejectedC);
```

Una promise può essere *completata con un valore* o *rifiutata con un errore*; qualsiasi sia il caso, verrà eseguito l'handler associato allo stato, messo precedentemente in coda dal metodo *then* della promise. *Inoltre, dato che then e catch ritornano promise, è possibile annidare "layer" di promise per elaborazione complesse e interdipendenti.*

Async/Await

Lo stile *async/await* si basa sulle *funzioni asincrone,* definite con la keyword *async*, e l'uso al loro interno della keyword *await*. Le *funzioni asincrone* possono essere viste come un *livello di zucchero sintattico* al di sopra del *sistema delle promise*: si dice che *una funziona asincrona è basata su promise poiché ha come valore di ritorno una promise.*

```
async function hello() {
  return greeting = await Promise.resolve("Hello");
};

hello().then(alert);
```

All'interno di una funzione asincrona, la keyword *await* può essere usata davanti al richiamo di un'altra funzione asincrona per interrompere l'esecuzione del codice fino al completamento della promise: rende sincrono una parte della funzione asincrona.

Richieste HTTP in JS

Le *richieste HTTP* che esegue il browser durante la navigazione del web sono *sincrone*: il browser fa la richiesta di una pagina, aspetta l'arrivo della risposta e infine visualizza la pagina.

In JS le richieste HTTP vengono attraverso la metodologia AJAX (Asynchronous JavaScript and XML): ovvero *richieste HTTP asincrone.*

JavaScript ha a disposizione due metodologie per eseguire una richiesta http in Ajax:

- *XHTTPRequest*;
- *Fetch*;

XHTTPRequest

XHTTPRequest è un oggetto che ci permette di eseguire una richiesta HTTP. L'invio di una richiesta HTTP comporta i seguenti passi:

1. Si crea un oggetto di tipo XHTTPRequest;
2. Tramite il metodo *open()* dell'oggetto *XHTTPRequest* si specificano *l'endpoint e il verbo HTTP;*
3. *Si definisce la funzione handler nella proprietà dell'evento* onreadystatechange dell'oggetto *XHTTPRequest*, questa funziona avrà il compito di gestire la *risposta HTTP* ricevuta dal server;

```
const http = new XMLHttpRequest();
const url = 'https://jsonplaceholder.typicode.com/todos/';
http.open("GET", url);

//define a functions that will be executed when a request is completed
http.onreadystatechange = (e) => {
    if (this.readyState == 4 && this.status == 200) {
        console.log(http.responseText)
    }
}

//send the request
http.send();
```

4. *Lanciare la richiesta http con il metodo send()* dell'oggetto *XHTTPRequest;*

Fetch

Il *fetch* è una API recente che permette di *eseguire richieste HTTP* basandosi su un sistema di *promise*. Il *metodo fetch() accetta una serie di parametri:*

- *Endpoint*: l'url su cui eseguire la richiesta;
- *Options*: è un array opzionale che contiene le opzioni della richiesta. Di default il fetch esegue richieste GET, ma è possibile specificare in questo array il *verbo HTTP o degli headers o il body contenente i dati di un POST;*

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

Si denoti che le Promise ottenute da un fetch() non assumeranno lo stato di Rejected qualora la risposta http contenga un error status code, la Promise sarà rifiutata solo in caso di errori di rete o un qualunque problema che prevenga il completamento della richiesta. La Promise viene "risolta" non appena il server risponde con degli header: la property **ok** *dell'oggetto response sarà posto a* ***true*** *solo quando lo status code ricevuto sia nel rangr 200-299.*

PHP

Introduzione

PHP (*PHP: Hypertext Preprocessor)* è un *linguaggio di programmazione e scripting* interpretato, ovvero il codice per essere eseguito non è soggetto a compilazione bensì vi è un interprete che legge ed esegue il codice.

PHP è un linguaggio *dinamicamente tipizzato* (o *debolmente tipizzato),* ovvero il *tipo delle variabili* è valutato a *runtime*, ma con PHP 7 hanno aggiunto il supporto agli "Scalar Type Declarations" che non è attivo di default, è necessario un'istruzione per abilitare l'opzione *strict_type,* ponendola a 1.

PHP *è case-insensitive* per le *keyword dei costrutti, classi e funzioni* mentre è *case-sensitive* per *i nomi delle variabili.* Come nella maggior parte dei linguaggi, *ogni istruzione con un punto e virgola.*

PHP è specialmente adatto, usato e pensato allo sviluppo web, per questo *user-case possiamo dire che PHP viene interprato dal server Web, ovvero server-side.*

L'utilizzo di PHP permette lo sviluppo di *pagine web dinamiche*, ovvero *il contenuto visualizzato nel browser non è statico ma potrebbe variare a seconda dell'input dell'utente o dell'ambiente.*

Un codice PHP è sempre contenuto all'interno di *tag speciali:*

- *<?php:* tag di apertura di un *blocco di codice PHP*. È usato sia per il *PHP embeded sia per un file PHP puro;*
- *?>:* tag di chiusura di un *blocco di codice PHP.* È necessario per chiudere il blocco per il *PHP embeded, mentre è opzionale nei file PHP puro*;

PHP Embed vs PHP Puro

Il *codice PHP* lo si può trovare in:

```php
<!- embedded.php ->
<html>
<body>
Today is
<?php echo date("l"); ?>
</body>
</html>
```

- *codice HTML in una forma embeded:* il codice PHP è inserito in mezzo al codice HTML attraverso l'uso di un *tag* speciale. Alla *richiesta del file*, il web server manderà il *codice HTML* così com'è ma qualora vi sia del codice PHP, questo verrà prima dato in pasto *all'interprete PHP* che lo elaborerà e stamperà eventualmente il risultato sotto forma di elementi HTML;
    - Questa tecnica ci permette di scrivere le pagine web come si è sempre fatto e arricchirle di componenti dinamiche senza troppi sforzi;

```php
<!- pure.php ->
<?php
echo "<html>";
echo "<body>";
echo "Today is";
echo date("l");
echo "</body>";
echo "</html>";
```

- *file PHP puro:* si definisce *file PHP puro* un file con estensione *.php* che non presenta la *struttura del documento HTML* ma contiene essenzialmente *codice PHP* (può contenere codice che stampi elementi HTML tuttavia);
    - *Questo approccio risulta più lento del'embeding perché dovremmo scrivere tutto il codice HTML tramite funzioni di stampa ma è assai più potente poiché permette una maggiore complessità e livello di controllo;*

PHP Include

In PHP, come in molti altri linguaggi, il codice può essere sparso in più file al fine di avere un codice più organizzato e mantenibile.

PHP mette a disposizione *le keyword include e require* che permettono l'operazione di inclusione, un'operazione che copia il contenuto di un file php, HTML o testuale all'interno del file che esegue l'inclusione.

Il funzionamento di *include e require* sono *identici* eccetto per il comportamento in caso di *fallimento* (file non trovato):

- *Require:* produrrà un *errore fatale* e *interrompe l'esecuzione dello script;*
- *Include:* produrrà un *warning* ma non *interrompe l'esecuzione dello script;*

Di entrambe le keyword esiste la variante **_once* che permette di evirare l'inclusione multipla di un file.

Commenti nel codice PHP

PHP prevede *commenti a singola riga e commenti multi riga:*

```
// a one line comment
# one line comment too
/* this is a longer,
   multi-line comment
 */
```

Variabili nel codice PHP

In PHP le variabili *devono iniziare con un segno del dollaro ($)* seguito dal *nome della variabile,* che può essere composta da *lettere, numeri e underscore. Il nome della variabile non può iniziare con un numero.*

***PHP al contrario degli altri linguaggi non ha una sintassi per dichiarare una variabile, quest'ultima viene creata all'assegnamento di un valore*.**

Scope delle variabili PHP

In PHP è possibile definire una variabile in un qualsiasi punto dello script e a seconda della posizione può essere:

- *globale*: se definita al di fuori di una funzione, sarà accessibile solo al di fuori delle funzioni;
  - La keyword ***global* seguita da una variabile** *può essere usata all'interno di una funzione per "importare" una variabile con scope globale e quindi poterla usare;*

    ```
    global $x, $y;
    ```

  - *Le variabili globali accessibili anche dall'interno della* ***variabile superglobale $GLOBALS:***

    ```
    $GLOBALS['x']
    ```

- local: se definita all'interno di una funzione, sarà accessibile solo all'interno della funzione in cui è stata definita. Al termine dell'esecuzione della funzione la variabile viene eliminata;
- static: se definita con l'uso della keyword *static* all'interno di una funzione, sarà accessibile solo all'interno della funzione in cui è stata definita. Al termine dell'esecuzione della funzione la variabile non verrà eliminata, bensì manterrà il suo valore per i successivi richiami della funzione;

  ```
  static $x = 0;
  ```

Variabili costanti in PHP

Le *variabili costanti* in PHP si definiscono attraverso la funzione *define()* e oltre ad essere immutabili sono accessibili globalmente da ogni punto dello script.

$$\text{define}(name,\ value,\ case\_insensitive)$$

- *name*: specifica il *nome della* costante, *il nome non deve essere preceduto da un segno del dollaro e può iniziare solo con una lettera o underscore;*
- *value*: specifica il *valore assunto dalla costante;*
- *case-insensitive*: un flag che specifica se il *nome della costante debba essere case-insensitive, di default è posto su false;*

```
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
```

Variabili superglobali in PHP

In PHP vi sono alcune *variabili superglobali, accessibili da qualunque scope.* Tali variabili *contengono informazioni di varia natura e per diversi scopi*.

- $GLOBALS: *un array associativo* che contiene tutte le *variabili globali* definite dall'utente e le rende accessibili in qualsiasi punto del codice. Le *chiavi* sono rappresentate dai nomi delle variabili;

- $_SERVER: *un array associativo* che contiene *le informazioni sugli header della richiesta HTTP ricevuta, le path e la locazione degli scripts*;
- $_REQUEST: *un array associativo* che contiene *tutti i dati raccolti dal form HTML che ha lanciato la richiesta*;
- $_POST: *un array associativo* che contiene *tutti i dati raccolti dal form HTML che ha lanciato la richiesta con metodo POST*;
- $_GET: *un array associativo* che contiene *tutti i dati raccolti dal form HTML che ha lanciato la richiesta con metodo GET;*
- $_FILES: *un array associativo* che contiene *i dati relativi a un file uploadato mediante form HTML;*
- $_SESSION: *un array associativo* che contiene *tutti i dati della sessione utente;*
- *$_COOKIE:* : *un array associativo* che contiene *tutti i cookie legati alla pagina web;*

Tipi primitivi e conversione

PHP prevede come tipi di dati primitivi:

- *string:* concatenabili attraverso l'operatore di concatenamento "**.**";
- *integer:* un numero intero, una stringa che rappresenta un intero o un numero *float* possono essere *castati a integer =>* `$int_cast = (int)$x;`
- *float:* un numero decimale;
- *boolean:* un booleano vero o falso;
- *array;*
- *null*;
- *object*;
- *resource*;

Essendo PHP un linguaggio *dinamicamente tipizzato*, la conversione del tipo di una variabile è eseguita automaticamente durante l'esecuzione del codice, a meno che non si sia attivata la flag *strict_types*.

Echo vs Print

Le funzioni *echo e print()* sono le principali funzioni di *stampa* di PHP. Queste *funzioni sono essenzialmente simili con una piccola differenza:*

- *echo non ritorna nessun valore e può accettare più parametro separati da virgole;*
```
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
```
- *print()* ritorna il valore **1** e *accetta solo un parametro, ovvero la stringa da stampare;*

If, switch, while, for

I costrutti e il funzionamento degli *if condizionale, switch, ciclo while e ciclo for* sono paragonabili a quelli degli altri linguaggi. Si è preferito omettere tale parte.

Funzioni PHP

La definizione di una funziona PHP consiste nella keyword *function* seguita da:

```
function Test($a, $b = 50, &c) {
  ...
}
```

- il *nome* della funzione;
- una *lista di parametri* separati da virgole racchiusa in parentesi tonde;
  - Se un parametro è *preceduto da un &,* la variabile sarà passata *byReference;*
  - *È possibile definire un valore di default di un parametro assegnando un valore al parametro stesso;*
- il *corpo della funzione* racchiuso dentro parentesi graffe;

Array in PHP

In PHP un array viene definito attraverso la funzione *array()* che accetta come parametro una lista di dati separati dalla virgola.

In PHP abbiamo 3 tipi di array:

- *Indexed arrays*: array accessibili tramite index numerico;

```php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
```

- *Associative arrays*: array accessibili con index letterale, ovvero una stringa;

```php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
```

- *Multidimensional arrays*: matrici & co;

```php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

```php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
```

OOP in PHP

PHP abbraccia il paradigma OOP, è quindi possibile definire classi, classi derivate, classi astratte e interfacce.

Classi e Oggetti

Una classe è definita utilizzando la keyword *class* seguita dal *nome della classe* e una coppia di *parentesi graffe*. Le proprietà e i metodi vengono definiti all'interno delle parentesi graffe.

All'interno di una classe è possibile definire gli scope di proprietà e metodi in modo analogo al C++, ovvero: *public, private e protected.*

Per creare un'oggetto di una classe si utilizza la keyword *new*, come negli altri linguaggi. Per *accedere alle proprietà e metodi* di un'oggetti si utilizza la *notazione a freccia* (−>).

```php
class Fruit {
  // Properties
  public $name;
  public $color;
  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
$apple = new Fruit();
echo $apple->get_name();
```

PHP *implementa il concetto di this* come tutti gli altri linguaggi, ovvero il *this* all'interno di un metodo della classe viene bindato all'oggetto chiamante.

Costruttori e Distruttori

Il *costruttore e il distruttore* sono metodi speciali all'interno di una classe che vengono eseguiti rispettivamente *alla creazione dell'oggetto e alla distruzione dell'oggetto.*

Il *costruttore* viene utilizzato per *inizializzare uno stato iniziale* dell'oggetto.

```php
class Fruit {
  public $name;
  public $color;
  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}
```

Costanti nelle classi

Le variabili costanti di una classe sono *case-sensitive* e si definiscono con la keyword *const.*
Per accedere a tali costanti si utilizza l'*operatore di risoluzione dello scope (::)*:

```php
class Test {
  const T = "Thank you";
}
```

- Da dentro un metodo della classe in cui è definita la costante è possibile usare la keyword *self;*

```php
self::T;
```

- Da un qualsiasi punto del codice è possibile usare il nome della classe:

```php
Test::T;
```

Metodi statici e proprietà statiche nelle classi

Il concetto di *metodi statici e proprietà statiche* è analogo a quello degli altri linguaggi, *ovvero metodi o proprietà che non sono legate a un'istanza della classe e sono direttamente accessibili utilizzando* l'*operatore di risoluzione dello scope (**::**)*:

- Da dentro un metodo della classe in cui è definita la costante è possibile usare la keyword *self;*
- Da un qualsiasi punto del codice è possibile usare il nome della classe:

```
self::T;
Test::T;
```

Ereditarietà

I concetti di ereditarietà in PHP sono analoghi a quelli degli altri linguaggi come il C++: si usa la keyword *extends*, vi è il concetto di proprietà o metodo *protected* e vi è la possibilità di fare *override dei metodi (non vi è la* keyword override*, si definisce un nuovo metodo con lo stesso nome).*

Una differenza importante è che PHP supporta solo la *ereditarietà singola*, ovvero *una classe può estendere solo un'altra classe.*

PHP fornisce inoltre la keyword *final,* la quale:

- se precede la definizione della classe ne previene l'estensione;
- se precede la definizione di un metodo ne previene l'override;

Per poter accedere a *variabili constati, metodi statici o proprietà statiche* si utilizza la keyword *parent seguita dall'operatore di risoluzione dello scope*.

```
class Fruit {
  protected $name;
  public function __construct($name) {
    $this->name = $name;
  }
  final public function intro() {
    echo "The fruit is {$this->name}";
  }
}
class Strawberry extends Fruit {
  public function message() {
  $this->name= "Test";
    echo "Am I a fruit or a berry?";
  }
  # public function intro() {..} ERROR!
}
```

Traits

I traits PHP costituiscono la soluzione alla limitazione della *singola ereditarietà*: in un trait è possibile definire metodi *con uno scope* che possono essere usati da molteplici classi.

Un trait è definito utilizzando la keyword *trait* seguita dal *nome del trait* e una coppia di *parentesi graffe*. I metodi vengono definiti all'interno delle parentesi graffe.

Una classe può fare uso di *uno o più trait,* per specificare che una classe utilizza *un determinato trait* si utilizza la keyword *use* seguita dal *nome del trait*, in caso di trait multipli si separano i nomi con le virgole.

```
trait TraitName {
  // some code...
}
class MyClass {
  use TraitName;
}
```

Classi astratte e metodi astratti

I concetti di classe e metodi astratti sono analoghi a quelli degli altri linguaggi: si definisce *classe astratta* una classe che ha *uno o più* metodi astratti, si definisce *metodo astratto* un metodo che è stato solo dichiarato ma non definito. Una *classe astratta* non può essere istanziata, deve essere estesa per definire le parti lasciate in sospeso.

Per definire una classe o un metodo astratto si utilizza la keyword *abstract* che prevede la keyword *class o function*.

```
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}
```

Interfacce

Il concetto di interfaccia è analogo a quello degli altri linguaggi: un'interfaccia specifica quali metodi una classe dovrebbe implementare.

```php
interface Animal {
  public function makeSound();
}
class Cat implements Animal {
  public function makeSound() {
    echo "Meow";
  }
}
```

Un'interfaccia si differenzia da una classe astratta perché:

- Non può avere proprietà;
- Tutti i metodi devo essere astratti e con scope pubblico;

Per definire un'interfaccia si utilizza la keyword *interface* seguita dal nome dell'interfaccia, mentre per specificare che una classe usa un'interfaccia si utilizza la keyword *implements* seguita dal nome dell'interfaccia.

Namespace

Il concetto di *namespace* è analogo a quello di C++: viene usato per raccogliere sono un nome comune classi ed evitare possibili problemi di collisione dei *nomi delle classi.* Per dichiarare o usare un namespace si utilizza la keyword *namespace* seguita dal *nome della namespace*.

Validazione dei dati di un form in PHP

Una volta ottenuti i dati dalle *variabili superglobali (*$_REQUEST, $_POST, $_GET), PHP mette a disposizione un set di *filtri* atti a controllare la validità del dato.

I *filtri* messi a disposizione sono svariati e si specializzano nel controllo di un tipo di dato: email, numero, ip, ect. Per i dati di cui non è disponibile un filtro è possibile fare una validazione basata su *pattern regex*.

La validazione può quindi essere eseguita in due modi:

- Se si ha a disposizione un *filtro*: si utilizza la funzione *filter_var($data, $filter)* che accetta come primo parametro il dato da validare e come secondo parametro il filtro da utilizzare e ritorna un booleano;

```php
filter_var($email, FILTER_VALIDATE_EMAIL)
```

- Se non si ha a disposizione un *filtro*: si utilizza la funzione *preg_match($pattern, $data)* che accetta come primo parametro un *pattern regex* e secondo parametro il dato da validare e ritorna un booleano;

Cookie in PHP

In PHP la creazione o modifica di un cookie, un dato client-side, viene effettuata mediante la funzione:

```php
setcookie(name, value, expire, path, domain, secure, httponly);
```

Solo il parametro *name* è obbligatorio, *gli altri sono opzionali*.

Per poter leggere un cookie si utilizza la *variabile superglobale $_COOKIE*:

```php
echo "Value is: " . $_COOKIE[$cookie_name];
```

JSON in PHP

Per poter gestire il JSON, PHP mette a disposizione due *metodi:*

- *json_encode()*: una funzione che *converte* un *valore* in *formato JSON*;
- *json_decode():* una funzione che converte un'oggetto JSON in un *oggetto PHP o Array Associativo;*
  - *Il secondo parametro è opzionale e se posto a true verrà creato un array associativo;*

Sessioni in PHP

Le sessioni sono un potentissimo strumento che ci permettono di *immagazzinare informazioni* che devono essere usate *su più pagine web durante la navigazione*. Le informazioni vengono salvate nella *variabile superglobale $_SESSION e tramite quest'ultima è possibile leggerle o modificarle.*

Per usare una sessione è necessario eseguire il metodo session_start() prima del *tag html*. Per terminare una sessione è necessario eseguire il metodo *session_unset(), per rimuovere le variabili di sessione, e* il metodo *session_destroy().*

Lettura e scrittura di file in PHP

Per poter manipolare i file, PHP mette a disposizione svariate funzioni con diverse funzionalità

- *readfile*($path): legge tutto il contenuto del file, la path del file deve essere fornita come parametro;
- *fopen($path, $flag): è* una funzione più avanzata simile alla controparte C,accetta come primo parametro la path del file e come secondo parametro una flag che determina la modalità di apertura (lettura, scrittura, append, ect..) e restituisce un *handler del file*. Sull'handler è possibile eseguire:
  - *fclose():* per *chiudere l'handler*;
  - *fread($handler, $bytes):* legge dal file il quantitativo di byte specificato;
  - *fgets($handler):* legge una riga del file;
  - *fgetc($handler):* legge un singolo carattere del file;
  - *feof($handler):* controlla se il file è terminato, End-Of-File;
  - *fwrite($handler, $data)*: scrive nel file la stringa specificata.

File upload in PHP

Attraverso un *form HTML e in input di tipo file,* un utente potrebbe mandare al web server un *file*. In PHP la *variabile superglobale $_FILES* contiene tutte le informazioni sul file uploadato:

- *nome*
- *dimensione*
- *posizione temporanea del file sul server*
- *informazioni sullo stato dell'operazione di upload*

Lo sviluppatore, dopo i dovuti controlli di validità del file, dovrà quindi usare le funzioni I/O di PHP per spostare il file nella posizione voluta:

- file_exists($file): controlla se un file esiste;
- `move_uploaded_file($uploaded_tmpName, $newFile):sposta un file uploadato in nella sua nuova posizione;`