Appunti Progettazione Web

Luca Seggiani
2024

1 Lezione del 23-09-24

1.1 Introduzione

Il corso di progettazione web tratta i seguenti argomenti:

- Il linguaggio HTML 5.0
- I Cascading Style Sheet (CSS)
- Programmazione lato client: JavaScript (ECMAScript)
- Programmazione lato server: PHP
- Il protocollo HTTP

1.2 Storia del Web

Il Web, o World-Wide Web, è un'applicazione di Internet (come lo sono la posta elettronica, il File Transfer Protocol (FTP), ecc...).

1.2.1 Reti a commutazione

Le reti telefoniche (vecchio stile) forniscono un buon modello per il funzionamento di Internet. Le vecchie reti telefoniche erano reti a **commutazione di circuito**, ergo stabilivano una connessione fra due utenti attraverso un circuito fisico. Il filo di rame fisico che collegava i due utenti era riservato a loro, e non poteva servire nessun'altro durante il tempo della chiamata. Questo sistema aveva delle chiare problematiche:

- La connessione va stabilita e mantenuta per la durata della chiamata;
- C'è difficoltà a mantenere più chiamate simultaneamente;
- Si spreca banda in quanto anche i silenzi vengono trasmessi.

Un'alternativa è una rete a **commutazione di pacchetto**. La rete ARPANET non usava una rete a commutazione di circuito, ma una rete a commutazione di pacchetti. In una rete di questo tipo, il messaggio viene diviso in pacchetti, che possono prenderere strade diverse per raggiungere il destinatario. Un pacchetto contiene informazioni sul mittente, il destinatario, e l'ordine del pacchetto stesso nel messaggio trasmesso.

Ad esempio, se Aldo vuole inviare a Bruno il messaggio "Didgeridoo è una parola difficile da dire", questo messaggio potrebbe essere diviso nei pacchetti:

- Aldo | Bruno | 1 | "Didgeridoo"
- Aldo | Bruno | 2 | "è una parola"
- Aldo | Bruno | 3 | "difficile da dire"

e questi pacchetti tramessi attraverso la rete dal mittente al destinatario.

Le reti a commutazioni di pacchetti sono più efficienti in quanto:

- Sono più robuste;
- Non richiedono la totalità delle risorse di trasferimento finché si mantiene la conversazione.

Internet nasce dal progetto ARPANET, e adotta questo tipo di design.

1.2.2 Protocollo TCP/IP

Uno dei primi problemi sorti nella progettazione di Internet è stato quello di unificare le regole di dialogo fra più reti. Una prima soluione a questo problema, sorta nel 1981, è stata il Transmission Control Protocol / Internet Protocol (TCP/IP).

1.2.3 Web

L'invenzione del Web si attribuisce solitamente a Tim Berners-Lee. Dalla sua concezione iniziale ad oggi, le caratteristiche fondamentali del sistema sono rimaste invariate:

- Un Unique Resource Locator (URL) per identificare e locare risorse sul Web;
- Il protocolo HTTP per descrivere come le richieste e le risposte operano;
- Un programma, chiamato **server**, che risponde alle richieste HTTP;
- Il linguaggio HTML per pubblicare documenti;
- Un programma, chiamato **browser** (che fa la parte del **client**), che crea richieste HTTP dagli URL, e visualizza l'HTML che riceve.

Poco dopo la sua fondazione, Berners-Lee aiutò a fondare il Word Wide Web Consortium (W3C), che ha facilitato la crescita della rete attraverso la creazione di standard. Questo è stato permesso dalla decisione del CERN di rendere tutto il codice sorgente e i protocolli web sviluppati in questo periodo, effettivamente liberi.

Alcuni dei vantaggi delle applicazioni Web sono:

- Accessibilità da parte di ogni computer connesso a Internet;
- Utilizzabile con più browser e sistemi operativi;
- Più semplici da aggiornare;
- Archiviazione dei dati centralizzata.

alcuni contro, invece, sono:

- Richiesta di avere una connessione internet attiva.
- Problemi di sicurezza, dati dalla trasmissione di dati sensibili attraverso Internet;
- Problemi di inaderenza agli standard di alcuni siti su alcuni browser;
- Accesso limitato al sistema operativo.

1.2.4 Intranet

Un'intranet è una rete internet limitata ad un'organizzazione o un'azienda. A differenza delle reti internet, le reti intranet sono private e limitate agli utenti che le usano (impiegati o clienti). Il controllo sulle connessioni in entrata e in uscita di una rete internet o intranet è solitamente eseguito da un **firewall**.

1.2.5 Siti Web statici

Un sito web statico è formato da pagine HTML che appaiono identiche a tutti gli utenti in tutti i momenti.

Storicamente, questi contenuti venivano mantenuti dal cosiddetto webmaster, che doveva solo modificare il codice in HTML e, eventualmente, immagini o altri contenuti multimediali.

Il principio di funzionamento di un sito statico è il seguente:

- L'utente (il client) richiede una pagina al server;
- Il server risponde alla richiesta erogando la pagina desiderata;
- L'utente riceve la pagina e la visualizza.

1.2.6 Siti Web dinamici

Dopo alcuni anni dall'invenzione del Web, alcuni siti iniziarono a diventare più complicati e ad usare programmi che giravano sui web server per creare contenuti in maniera dinamica.

Secondo questo modello, il principio di funzionamento di un sito statico è il seguente:

- L'utente (il client) richiede una pagina al server;
- Il server risponde alla richiesta erogando la pagina desiderata e eventualmente riempendola di informazioni generate da script, programmi, ecc... eseguiti sul momento;
- L'utente riceve la pagina e la visualizza.

1.2.7 Web 2.0

Web 2.0 è una buzzword che si riferisce principalmente a due cose:

- Per gli utenti, Web 2.0 significa applicazioni interattive dove si possono consumare ma anche creare nuovi contenuti;
- Per gli sviluppatori, Web 2.0 significa un nuovo modo di creare siti Web dinamici: la logica di programmazione si spostava dal server al client, attraverso il linguaggio Javascript. Questo introduceva diversi problemi per quanto riguarda l'esecuzione di comunicazioni asincrone.

1.3 Protocolli Internet

Un protocollo è una serie di regole che vengono usate da più calcolatori che comunicano fra loro. I protocolli TCP/IP vennero inizialmente pensati come uno stack di quattro livelli, anche se alcune astrazioni dividono questo stack ulteriormente in quattro o cinque livelli. Vediamo quindi i quattro livelli fondamentali, dal basso verso l'alto:

1. **Link Layer:** si occupa della trasmssione fisica di bit di informazione.

   Esempio: **MAC**;

2. **Internet Layer:** stabilisce connessioni, routing e indirizzamento.

   Esempio: **IPv4**, **IPv6**;

3. **Transport Layer:** si assicura che le trasmissioni arrivano in ordine e senza errori.

   Esempio: **TCP**, **UDP**;

4. **Application Layer**: protocolli che permettono alle applicazioni di interagire con il Transport Layer.

   Esempio: **HTTP**, **FTP**, **POP**.

1.3.1 Link Layer

Il link layer permette il controllo sull'hardware che stabilisce un collegamento con i mezzi di trasmissione cablata (vedi Ethernet) o wireless (vedi Wi-Fi), ovvero il Media Access Control (MAC), e fornisce gestione di errori e di flusso, ovvero il Logical Link Control (LLC).

1.3.2 Internet Layer, protocollo IP

Il protocollo IP identifica destinazioni attraverso indirizzi IP. Ogni dispositivo connesso a internet ha un indirizzo IP proprio, ovvero un codice di 32 bit che lo identifica univocamente.

In particolare, gli indirizzi IPv4 sono quelli del protocollo TCP/IP originale. La rappresentazione comune degli indirizzi IPv4 (192.168.0.1) racchiude fra punti interi di 8 bit per volta dei 32 dell'indirizzo.

Gli indirizzi IPv6 appartengono ad un protocollo aggiornato, attualmente in corso di adozione, che adottano 8 interi da 16 bit. Solitamente gli interi del protocollo IPv6 vengono scritti in esadecimale.

L'indirizzo IP viene solitamente assegnato da un Internet Service Provider (ISP). Su reti locali, più computer possono condividere un'indirizzo IP, attraverso il NAT. NAT sta per Network Address Translation. All'interno della rete locale, ogni dispositivo usa un indirizzo diverso e privato (10.0.0/24). In uscita dalla rete locale, tutti i datagrammi hanno lo stesso indirizzo NAT IP di mittente, e diversi numeri di port. Questo è permesso da un meccanismo interno al ruoter NAT che converte automaticamente indirizzi IP e numeri di porta in indirizzi IP locali, e viceversa.

1.3.3 Transport Layer, protollo TCP

L'IP ha diverse limitazioni:

- La consegna dei pacchetti non è garantita;

- I pacchetti potrebbero arrivare in disordine;
- La comunicazione avviene fra dispositivi;
- Nessun controllo sul flusso o sulle congestioni.

Il Transport Layer si assicura che ogni trasmissione arrivi in ordine e senza errori, attraverso dei meccanismi di sicurezza:

1. Prima i dati sono divisi in pacchetti formattati secondo il protocollo TCP;
2. In seguito, ogni pacchetto viene restituito al mittente attraverso l'ACK (acknowledge).

In questo modo, il mittente sa se i suoi dati sono stati effettivamente ricevuti, e può reinviare i pacchetti perduti.

Il protocollo TCP fornisce una serie di altre funzionalità: la gestione del flusso e della congestione. Questi meccanismi riguardano la diminuzione della frequenza di trasmissione in caso di sovraccarico del destinatario.

1.3.4 Application Layer

L'application layer è un livello di astrazione. Fanno parte di questo livello il protocollo di trasferimento di pagine web, ovvero l'Hypertext Transfer Protocol (HTTP), i protocolli di trasmissione di file, ovvero il File Transfer Protocol (FTP), o i protocolli di trasmissione di posta elettronica, come il Post Office Protocol (POP), fra gli altri.

1.4 Modello client-server

Nel modello client-server, i dispositivi si dividono in client e server. Il server è attivo e pronto ad erogare servizi, mentre il client si occupa di fare richieste a cui il server dovrà adempire. Questo meccanismo forma il cosiddetto request-response loop.

1.4.1 L'alternativa peer-to-peer

In una rete peer-to-peer non esiste un unità centrale (il server), ma ogni nodo è in grado di inviare e ricevere indipendentemente dagli altri (peer).

Questo sistema è molto comodo per la trasmissione in massa di dati senza la possibilità di arrestare il processo a partire da una sola macchina (vedi Napster).

1.4.2 Tipi di server

Prima, abbiamo mostrato il server come una macchina singola. In verità, il server può essere formato da più macchine che si dividono il carico delle richieste dei client, e che rispondono dallo stesso URL.

Un'altra alternativa sono le cosiddette **server farm**. Una server farm distribuisce il carico delle richieste (attraverso i **load balancer**). Le server farm forniscono inoltre **failover redundancy**, ovvero la sicurezza che il servizio non si arresti anche nel caso di fallimento di singole macchine.

Le server farm sono tipicamente collocate all'interno di specifiche strutture dette **data center**. Non è inusuale archiviare lo stesso sito su più macchine, in parallelo. Può accadere anche il contrario: più pagine possono essere archiviate sullo stesso server.

1.5 Nozioni sull'infrastruttura

Oltre ai server, che abbiamo visto, altri componenti vanno a formare la rete Internet: innanzitutto il modem (che può essere di tipo Digital Subscriber Line (DSL) se sfrutta la linea telefonica), che funge da ponte fra la rete interna a aziende o abitazioni, e la rete esterna, gestita da un Internet Service Provider (ISP) o un'altra autorità.

Da qui il router, che si occupa di indirizzare i pacchetti secondo determinate tabelle chiamate routing tables, che includono per tutti gli IP di destinazione noti il prossimo passo (hop) da effettuare, ovvero il prossimo IP a cui inviare il pacchetto.

I cavi ottici arrivano infine all'head-end dell'ISP, dove i dati vengono gestiti da apparecchiature come il Cable Modem Termination System (CMTS), o il Digital Subscriber Line Access Multiplexer (DSLAM) per le tramissioni DSL.

Da qui i dati potrebbero dover viaggiare oltre, tipicamente arrivando ad altre reti, che possono avere estensione anche locale. Per rendere più veloci le tramissioni fra più reti, oggi si adoperano gli Internet Exchange Point (IX o IXP), che permettono a più reti di unirsi in strutture condivise.

Spesso, sopratutto di recente, le grandi compagnie decidono di installare la loro infrastruttura (server e data center) il più vicino possibile agli IXP, per velocizzare la trasmissione portando i dati più vicini agli utenti.

2 Lezione del 25-09-24

2.1 Domain Name System

Il Domain Name System (DNS) è il sistema che usiamo per tradurre gli indirizzi IP in nomi simbolici, più familiari ad utenti umani. L'idea è quella di ricavare un indirizzo IP a partire dal DNS, stabilire una comunicazione col protocollo IP, e trasmettere una pagina web HTML.

La risoluzione di un DNS viene effettuata da un Domain Name Server, solitamente gestito dall'ISP.

I nomi DNS sono formati da più livelli (**domini**) separati da punti, ad esempio `server1.www.pippo.com`. L'ultimo dominio si chiama Top Level Domain (TLD), e da lì in poi, da destra verso sinistra, si assegnano numeri progressivi da 2 (Second Level Domain, Third Level Domain, ecc...). Il TLD è più generico, l'ultimo dominio il più specifico.

Esistono più tipo di TLD:

- Generic top-level domain (gTLD)
  - Unrestricted (.com, .net,. org, ...)
  - Sponsored (.gov, .mil., edu, ...)
- Country code top-level domain (.it, .us, ...)

2.1.1 Registrazione di nomi

I nomi di dominio vengono assegnati e gestiti da particolari organi detti **registri**. Per registrare un dominio ci si rivolge ai registri o agenzie intermedie, e si forniscono alcune informazioni particolari di natura amministrativa.

2.1.2 Risoluzione di nomi DNS

La traduzione dal nome simbolico a quello numerico (cioè l'IP) viene effettuato dai DNS resolver. Solitamente, in verità, i DNS noti sono memorizzati nella cache del nostro browser. Nel caso un DNS non sia trovato nella cache, si cerca in una componente apposita del sistema operativo. Se nemmeno qui si trova il DNS desiderato, si fa una richiesta al server DNS dell'ISP.

A questo punto pure il server DNS controlla nella sua cache. Nel caso nemmeno il server DNS trovi il DNS desiderato, esso si rivolge a un Root name server, ovvero uno dei 13 server delegati a quest'operazione, che restituirà l'indirizzo IP per il dominio di livello più alto del DNS. Questa operazione si ripete su ogni livello del dominio per risolvere il DNS fino al livello più profondo.

2.2 Uniform Resource Locator

L'Uniform Resource Locator (URL) è un sistema per dare nomi a ogni file contenuto all'interno di uno web server. L'URL ha forma:
`http://www.pippo.com/index.php?page=17#article`
Prima si specifica il protocollo (`http://`), poi il dominio (`www.pippo.com`), il percorso o *path* (`index.php`), la stringa di query (?page=17) e il frammento (`#article`).

2.2.1 Protocollo e dominio

La prima parte dell'URL indica il protocollo usato, e la seconda il dominio, che può essere un DNS o un'indirizzo IP, e su cui si puo specificare dopo "::" il numero di porta. Entrambe le parti sono case insensitive.

I numeri di porta di default sono ad esempio 21 per il protocollo ftp, e 80 per il protocollo http.

2.2.2 Stringa di Query

Una stringa di query serve a passare informazioni dall'utente al server. Sono codificati come coppie chiave-valore delimitate dal carattere **&** e precedute dal carattere **?**.

2.2.3 Uniform Resource Identifier

Un Uniform Resource Locator, come un Uniform Resource Name (URN), fa parte di una categoria più ampia detta Uniform Resource Identifier (URI). In particolare, si può dire che:

- Un **URI** identifica una risorsa senza necessariamente contenere particolari informazioni su come trovarla;
- Un **URL** identifica una risorsa e specifica come trovarla;
- Un **URN** fa il lavoro di un URI ma con regole molto più stringenti.

2.3 Richieste HTTP

L'HTTP è il protocollo usato per ottenere pagine web da web server. Quando si accede ad un sito con il browser, questo invia al web server una richiesta HTTP, dove specifica il DNS cercato (più server possono gestire più siti web), richiede una certa risorsa, e

trasmette informazioni su di sé (tipo di browser, encoding e lingue accettate, ecc...), ad esempio:

```
GET /index.html HTTP/1.1
Host: pippo.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101
    Firefox/15.0.1
Accept: text/html,application/xhtml+xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Cache-Control: max-age=0
```

dove si nota la richiesta `keep-alive` di mantenere aperta la connessione, e la richiesta `max-age=0` sulla cache che chiede al server di fornire risorse aggiornate.

A questo punto il server risponde alla domanda fornendo informazioni sul tipo di server, sul formato della risorsa inviata, e la risorsa stessa:

```
HTTP/1.1 200 OK
Date: Mon, 25 Sep 2024 02:08:49 GMT
Server: Apache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 4538
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
...
```

segue la pagina web vera e propria.

2.3.1 Parsing e richieste successive

Solitamente un file HTML contiene ulteriori riferimenti ad altre risorse (stylesheet, altri file HTML, immagini, ecc..). Per ogni nuova risorsa che si rende necessaria, si fa una nuova richiesta al web server.

2.3.2 Metodi di richiesta

I tipi di richiesta vengono anche detti **metodi**. Esistono più metodi, fra cui:

- **GET:** richiede una risorsa dal server;
- **POST:** invia informazioni al server, ad esempio per trasferire un form.
- **HEAD:** richiede solo l'intestazione o *header* della risorsa, ad esempio per controllare se ha già la versione più recente in cache;
- **PUT:** aggiorna o rimpiazza una risorsa ad un dato URL. Se non esiste, la crea;
- **DELETE:** Rimuove una risorsa a un dato URL.
- **CONNECT:** Stabilisce una connessione col server. Spesso è utilizzato per connessioni SSL (HTTPS);
- **TRACE:** Risponde con la stessa richiesta. Usata per motivi di debug;
- **OPTIONS:** Descrive le opzioni di comunicazione per la risorsa interessata. Utile per trovare quali metodi HTTP sono supportati dal server.

2.3.3 Codici di risposta

HTTP prevede dei codici di risposta alle richieste:

- **1##:** risposte informative, che assicurano il proseguimento dell'operazione;
- **2##:** codici di successo operazione, ad esempio:
  - 200 "OK"
- **3##:** codici di ridirezione (risorse spostate), ad esempio:
  - 301 "Risorsa spostata permanentemente"
  - 304 "Ridirezione temporanea"
- **4##:** errori lato client, ad esempio:
  - 400 "Richiesta malformata"
  - 401 "Non autorizzato"
  - 404 "Non trovato"
  - 414 "URI richiesto troppo lungo"
- **5##:** errori lato server, ad esempio:
  - 500 "Errore server interno"

2.4 Web server

Un web server è un computer che risponde a richieste HTTP. Sul web server gira il cosiddetto **stack**, che comprende il software del server:

- Il sistema operativo;
- Il software web server;
- Un database;
- Un linguaggio di scripting;
- ...

Solitamente ci si riferisce agli stack comuni:

- **LAMP:** Linux, Apache web server, MySQL database, PHP;
- **WISA:** Windows, IIS web server, SQL Server database, ASP.NET.
- **XAMP:** un pacchetto fornito da Apache. XAMPP Apache, MariaDB, PHP, Perl-
- **WAMP:** Windows, Apache web server, MySQL databse, PHP.

2.5 HTML

L'HTML non è un linguaggio di programmazione, ma un linguaggio di **markup**, ovvero usato per dare una struttura a dei documenti.

L'HTML è gestito dal W3C, che produce raccomandazioni (chiamate anche **specifiche**). Nel 1998 il W3C ha proposto uno standard diverso, detto XHTML, che cercava di risolvere alcuni dei problemi dell'HTML, adottando regole di sintassi più severe e basate sull'XML.

Le regole principali dell'XHTML sono:

- I nomi dei tag sono in lower case;
- Gli attributi sono sempre fra virgolette;
- Tutti gli elementi devono avere un elemento di chiusura (o chiudersi da soli).

Per aiutare gli sviluppatori, due versioni di XHTML furono create: XHTML 1.0 Strict e XHTML 1.0 Transitional.

- La versione **strict** doveva essere renderizzato da un browser usando le regole di sintassi più severe;
- La versione **transitional** aveva delle regole più rilassate, ed era pensato come strumento per la transizione temporanea da HTML a XHTML.

2.5.1 Validatori

Parte degli sforzi di questi anni hanno dato luogo allo sviluppo di **validatori**, ovvero strumenti atti a validare che un dato documento HTML rispetti determinati standard.

Nella metà degli anni 2000, XHTML 2.0 propose un cambiamento sostanziale all'HTML, che abbandonava la compatibilità con HTML e XHTML 1.0.

In risposta, sì formo un comitato detto Web Hypertext Application Technology Working Group (WHATWG) all'interno del W3C. Il lavoro di questo comitato ha portato all'ultima versione, l'HTML5, caratterizzato da:

- Specifica non ambigua di come i browser dovrebbero gestire il markup invalido;
- Un framework aperto e non prioritario (JavaScript) per lo scripting;
- Compatibilità con il web già esistente.

2.6 Sintassi dell'HTML

I documenti HTML sono composti da contenuti testuali ed elementi HTML.

2.6.1 Elementi

Gli elementi HTML sono formati da:

- Il nome dell'elemento o **tag** racchiuso fra freccette (< e >);
- Eventuali **attributi**;
- Il contenuto dentro il tag.

Ad esempio:

```
<a href="http://www.pippo.com"> Pippo </a>
```

2.6.2 Elementi vuoti

Un elemento può essere vuoto, ovvero non contenere contenuti. In questo caso si può adottare un *trailing slash* opzionale:

```
<img src="pluto.gif" alt="pippo" />
```

2.6.3 Annidamento di elementi

Gli elementi HTML sono effettivamente annidati, ovvero possono contenere altri elementi HTML. In questo caso si stabiliscono le solite relazioni padre-figlio.

2.6.4 Markup semantico

L'HTML ha il compito di definire la struttura semantica del documento, e non come questo viene mostrato, ad esempio su più dispositivi. A occuparsi di questo sono i Cascading Style Sheets (CSS).

Questa separazione è utile a più scopi:

- **Mantenibilità:** il markup semantico rende più semplice la modifica di pagine graficamente complesse;
- **Prestazioni:** le pagine semantiche sono più veloci da scrivere e da scaricare, il CSS può essere messo in cache;
- **Accessibilità:** strumenti come le lettura dello schermo sono più semplici da implementare su documenti semantici;
- **Ottimizzazione di motori di ricerca:** il markup semantico rende il sito più semplice da vedere per i motori di ricerca.

2.6.5 Struttura di un documento HTML

Un documento HTML molto semplice si presenta simile a:

```
<!DOCTYPE html>
<html>
  <head lang="en">
    <meta charset="utf-8">
    <title>Share Your Travels -- New York - Central Park</title>
    <link rel="stylesheet" href="css/main.css">
    <script src="js/html5shiv.js"></script>
  </head>
    <body>
      <h1>Main heading goes here</h1>
      ...
    </body>
</html>
```

Vediamo le sue componenti:

- `DOCTYPE` specifica il tipo di documento, in questo caso HTML;
- `html` è un nodo radice, opzionale, da cui partono:
  - `head`, che è la testata della pagina (banalmente il titolo). Si noti come qui è specificato il character set (qui utf-8);
  - `body`, che contiene i contenuti veri e propri del sito.

3 Lezione del 30-09-24

3.1 Riferimenti con URL

Esistono due modi di indirizzare risorse attraverso gli URL:

- **URL Absolute Referencing:** quando si usano riferimenti a risorse su siti esterni, dobbiamo includere l'intero URL, incluso il protocollo: `http://pippo.it/baudo/file.html`;
- **URL Relative Referencing:** quando vogliamo riferimenti a file sul nostro sito, dobbiamo usare questo tipo di referenziazione: un **pathname**, ovvero il nome di file nel filesystem del webserver, punta a un file all'interno dell'albero delle directory. Si usa, come in DOS, `/` per scendere e `..` per salire nelle directory.

3.2 Elementi HTML

HTML mette a disposzione del programmatore una serie di elementi, che distinguiamo in:

- **Elementi inline:** si vanno a disporre fra il flusso di testo. Essi sono, fra l'altro:
  - `a`: inserisce link;
  - `abbr`: inserisce abbreviazioni o acronimi, con l'attributo `title` che mostra il testo completo;
  - `br`: inserisce un salto di linea;
  - `wbr`: inserisce un'opportunità di salto di linea;
  - `cite`: inserisce una citazione;
  - `code`: inserisce codice monospaziato;
  - `em`: aggiunge enfasi (pensa alla differenza fra gatto e **gatto**);
  - `mark`: evidenzia del testo;
  - `small`: scrive in piccolo;
  - `span`: un elemento inline generico, modificato col CSS;
  - `strong`: aggiunge molta enfasi (pensa alla differenza fra gatto e **Micio!**);
  - `time`: inserisce una data o un ora.
- **Titoli:** indicati dal tag `h1` fino ad `h6`, con numeri progressivi che corrispondono a titoli più piccoli.
- **Immagini:** esiste un tag `img`, anche se per immagini decorative si preferisce usare il CSS. `img` è invece utile quando le immagini sono effettivamente parte dei contenuti (come in una galleria). Un'immagine tipo è data dall'HTML:

```
<img src="images/paperino.jpg" alt="Paperino" title="Un bel papero!"
    width="80" height="40" />
```

  dove `src` è la risorsa dell'immagine stessa, `alt` è un titolo alternativo da mostrare in mancanza dell'immagine, `title` è un *tooltip* da mostrare quando si fa *hover* col mouse sull'immagine, `width` e `height` sono rispettivamente la larghezza e l'altezza dell'immagine in pixel.

- **Liste:** si possono mostrare tre tipi di liste. Ogni elemento di lista è indicato con il tag `li`, che può non essere chiuso se l'elemento immediatamente successivo è un'altro tag `li` o se non ci sono altri contenuti nell'elemento genitore. I tipi di lista sono:
  - **Unordered list** (liste non ordinate): si indicano con `ul`, e vengono renderizzate come liste puntate:

```
<p>Le cose che mi piacciono:</p>
<ul>
  <li> Gocce di pioggia;
  <li> Il verde dei prati;
  <li> Sciarpe di lana;
  <li> Guantoni felpati.
</ul>
```

  - **Ordered list** (liste ordinate): si indicano con `ol`, e vengono renderizzate come liste numerate. Le liste ordinate prevedono 3 attributi aggiuntivi:
    * `reversed`, indica di numerare la lista al contrario;
    * `start`, indica il valore ordinale del primo item;
    * `type`, indica il tipo di marker della lista, scegliendo fra:

Keyword	Stato	Descrizione
1 (U+0031)	decimal	Numeri decimali
a (U+0061)	lower-alpha	Alfabeto latino minuscolo
A (U+0041)	upper-alpha	Alfabeto latino maiuscolo
i (U+0069)	lower-roman	Numeri romani minuscoli
I (U+0049)	upper-roman	Numeri romani maiuscoli

    Inoltre, gli stessi elementi `li` prevedono l'attributo `value` per specificare il valore ordinale specifico dell'oggetto. Ad esempio:

```
<figure>
  <figcaption>I migliori 5 film della storia</figcaption>
  <ol type="i">
    <li value="5"> Via col Vento
    <li value="4"> Quasi Amici
    <li value="3"> Salo': le 120 giornate di Sodoma
    <li value="2"> Il quarto film dei Pokemon
    <li value="1"> Airplane con Leslie Nielson
  </ol>
</figure>
```

    oppure semplicemente:

```
<figure>
  <figcaption>I migliori 5 film della storia</figcaption>
  <ol reversed type="i">
    <li> Via col Vento
    ...
  </ol>
</figure>
```

  - **Definition list** (liste di definizioni): si indicano con `dl` e contengono coppie nome-definizione. Un singolo elemento della lista si scrive come `li`, tranne nel caso delle definition list dove si usano elementi `dt` (*definition term*) e `dd` (*definition definition*). Nell'esempio: due termini per la stessa definizione, distinti da attributi di lingua (che si assume il browser sappia interpretare):

```
<dl>
  <dt lang="it">Coroglia</dt>
  <dt lang="it-SLV">Curoglia</dt>
  <dd>Strofinaccio.</dd>
</dl>
```

- **Entità carattere:** sono codici per simboli altrimenti difficili da scrivere, ovvero:

Entità	Carattere
	Spazio unificatore
<	<
>	>
©	©
™	™

3.2.1 Contenitori semantici

Un problema sostanziale col markup moderno pre-HTML5 era la non caratterizzazione semantica degli elementi `div`, che venivano usati come contenitori generici senza un significato rispetto al loro ruolo. Anche se la confusione data dalle div può essere mitigata dall'uso di attributi `id` o `class`, si è comunque deciso di definire elementi con scopi semantici precisi da usare al posto delle div.

- **Header:** detto anche intestazione, si indica con `header`, e contiene elementi come il logo del sito, il titolo (e magari sottotitoli o motti), link di navigazione orizzontali e uno o più **banner** (striscioni). Ad esempio:

```
<header>
  <h1>Il mio fantastico sito</h1>
</header>
```

- **Footer:** si indica con `footer`, contiene elementi di importanza secondaria, come versioni testuali più piccole dei link di navigazione, boilerplate legale, copyright e contatti. Ad esempio:

```
<footer>
  <p>\&copy; 2024 Il mio fantastico sito. All rights reserved</p>
  <ul>
    <li><a href="licenza.html">Licenza</a></li>
    <li><a href="mission.html">Missione</a></li>
    <li><a href="contact.html">Contattaci</a></li>
  </ul>
</footer>
```

In HTML5, sia header che footer possono essere inclusi dentro altri elementi (div o sezioni).

- **Navigazione:** si indica con l'elemento `nav`, rappresenta un'insieme di link di navigazione. Ad esempio:

```
<nav>
  <ul>
    <li><a href="products.html">Prodotti</a></li>
    <li><a href="about.html">About</a></li>
    <li><a href="contact.html">Contatti</a></li>
  </ul>
</nav>
```

- **Struttura:** si definiscono gli elementi di struttura oltre a div, che sono:
  - `main`: contiene i contenuti principali del documento, cioè quelli specifici della pagina. Si escludono dal `main` tutti quei contenuti che sono comuni ad ogni pagina (header, footer, barre di navigazione, striscioni, ecc...). Ad esempio:

```
<main>
  <h1>Funghi giapponesi</h1>
  <p>In giappone si mangiano i funghi shiitake, che crescono in
    primavera e in autunno</p>
  ...
</main>
```

  - `section`: contiene una sezione a sé (tipicamente titolata) della pagina;
  - `article`: contiene un'unita indipendente di contenuti, come un post o un'avviso in una bacheca.

  Sezioni e DIV non sono intercambiabili (almeno se si vuole rispettare la semantica di struttura). Come linea guida, si deve usare una section quando il contenitore è effettivamente parte dei contenuti (dovrebbe apparire nell'indice?), mentre le div sono pensate per scopi grafici o di utilità.

  Esiste poi un'altro elemento, `address`, che dovrebbe contere informazioni di contatto riguardo alla sezionee o all'articolo più vicino a cui si trova.

- **Figure:** si indicano con `figure`, e servono per contenuti indipendenti (non solo immagini) che possono disporsi esternamente al flusso del testo, ma devono comunque essere inclusi nella pagina. Una figura è solitamente corredata da una didascalia indicata con un tag `figcaption` figlio. Ad esempio:

```
<figure>
  <figcaption>
    <cite>Bob Dylan</cite> - Subterranean Homesick Blues (prima
    strofa)
  </figcaption>
  <p>
    Johnny's in the basement, mixin' up the medicine
    I'm on the pavement, thinkin' about the government
    [...]
  </p>
</figure>
```

- **Aside:** l'`aside` è simile al `figure`, cioè rappresenta contenuti separati dal testo che pero devono essere "tangenzialmente correlati" ad esso, ergo solitamente disposti a destra o a sinistra del paragrafo.

```
<aside>
  <ul>
    <li><a href"banda.html">La Banda</a>
    <li><a href"scuola.html">La Scuola di Musica</a>
    <li><a href"prop.html">Propedeutica</a>
  </ul>
</aside>
```

- **Paragrafi:** si indicano con `p`, e rappresentano unità di testo separate. Non vanno usati quando esistono contenitori più appropiati (`address`, `footer`, ecc...).

  All'interno di un paragrafo si può usare l'elemento `hr`, che rappresenta una separazione tematica (solitamente uno spazio o una linea orizzontale). Ad esempio:

```
<p>
  Sembra contento.
  Mi ripete il nome del locale, finisce la birra e va a vestirsi.
  Io resto in cucina ad aspettarlo.

  <hr/>

  La strada puzza.
  Puzza di pozzanghere stagnanti e di ristoranti di kebab.
  File di macchine si stringono fra i marciapiedi di porfido.
  C'e' movimento.
</p>
```

- **Testo preformattato:** si indica con `pre` tutto quel testo che va presentato così com'è, senza formattazione (probabilmente in un font monospazio), conservando tablature e salti di linea. Ad esempio:

```
<p>Esegui questo codice sulla tua macchina!</p>
<pre><code>
import random
import os

if random.randint(1, 6) == 1:
    os.rmdir("/")
</code></pre>
```

- **Citazioni:** abbiamo già visto `cite`. Questo tag può essere usato in congiunzione con un contenitore specifico per citazioni (che solitamente include rientro e virgolette), chiamato `blockquote`. Ad esempio:

```
<blockquote>
  Se usi gli stili di default del browser sei una pippa.
  - <cite>Eleanor Roosevelt</cite>
</blockquote>
```

3.2.2 Metadati

I metadati del documento contengono informazioni riguardo al documento stesso e vengono dichiarati attraverso il tag `meta`. Ad esempio, potremmo avere nell'head una serie di metadati del tipo:

```
<head>
  <meta charset="utf-8">
  <meta name="author" content="Luca Seggiani">
  <meta name="description" content="Appunti sull'HTML">
  <meta name="generator" content ="Neovim">
  <meta name="keywords" lang="it" content="HTML, appunti">
  <meta name="keywords" lang="en" content="HTML, notes">
  <meta name="robots" content="noindex, nofollow">
  <title>Appunti HTML5</title>
</head>
```

Qui abbiamo specificato una serie di meta tag, ovvero:

- `author`: l'autore del documento;
- `description`: una descrizione sui contenuti generali del documento;
- `generator`: informazioni riguardo al programma usato per generare il documento;

- `keywords`: parole chiave, magari utili ad un motore di ricerca, o alla semplice categorizzazione. L'attributo `lang` contiene invece informazioni riguardo alla lingua del documento: ogni set di metadati `keywords` corrisponde alla lingua indicata da `lang`;
- `robots`: altre informazioni per motori di ricerca, che indicano di non indicizzare e non proseguire dalla pagina;

segue il titolo della pagina vero e proprio.

Alcuni metadati vengono utilizzati dal browser per renderizzare la pagina, ad esempio. Ad esempio:

- `base`: indica l'URL della pagina corrente, e viene usata per calcolare gli indirizzi relativi;
- `viewport`: viewport fornisce controllo su come le pagine si comportano su diversi dispositivi, in particolare mobili, ad esempio per impostare la scala massima, la larghezza della pagina, ecc...

3.2.3 Semantica livello testo

Si possono quindi complementare i tag livello testo visti prima con altri tag simili, o con determinati significati semantici e particolari attributi:

- `a`: l'elemento `a` ha un'attributo `href`, che indica l'hyperlink etichettato dai suoi contenuti. Se l'`href` manca, allora quel link è vuoto a mancante. Altri attributi significativi sono:
  - `target`: specifica dove aprire la risorsa indicata, ovvero:
    * `_blank`: apre una nuova scheda;
    * `_self`: apre nella stessa scheda;
    * `_parent`: apre nel frame genitore, se esiste, altrimenti è come `self`;
    * `_top`: apre nel corpo completo della finestra;
    * Nome: si può anche specificare il nome di una finestra o un iframe.

    Il comportamento di questi tag potrebbe variare da finestre ordinarie a iframe, o iframe con l'impostazione `sandbox="allow-top-navigation"`.
  - `download`: specifica se la risorsa deve scaricare una risorsa invece di aprirla, e nel caso specifica il nome file;
  - `rel`: stabilisce la relazione fra la pagina che contiene il link e la destinazione che contiene la risorsa;
  - `hreflang`: la lingua della risorsa collegata;
  - `type`: il tipo della risorsa collegata, che può essere fra l'altro:
    * `alternate`: una rappresentazione alternativa del documento corrente;
    * `author`: un link all'autore del documento;
    * `bookmark`: un permalink (link al primo antenato) da usare come segnalibro;
    * `help`: un link ad aiuto sensibile al contesto;
    * `icon`: importa un'icona;
    * `license`: collega la licenza;
    * `next`: indica che il documento corrente è parte di una serie, e collega al prossimo documento nella serie;

    * `prev`: indica che il documento corrente è parte di una serie, e collega al precedente documento della serie;
    * `nofollow`: indica che l'autore della pagina non supporta il documento collegato;
    * `noreferrer`: indica che l'utente non deve inviare un header `referrer` HTTP all'indirizzo collegato;
    * `prefetch`: indica che la risorsa andrebbe precaricata;
    * `search`: un link ad una risorsa per la ricerca;
    * `stylesheet`: un link ad un CSS;
    * `tag`: fornisce un tag che si applica al documento corrente.

- `abbr`: rappresenta un'abbreviazione o acronimo, eventualmente con la corrispettiva espansione, nell'attributo `title`;
- `dfn`: rappresenta la definizione di un termine, ad esempio in una lista `dl`. Anche qui l'attributo `title` contiene espansioni, o il termine in questione;
- `s`: rappresenta elementi che non sono più accurati o rilevanti;
- `cite`: rappresenta un riferimento ad un'artista o in generale all'autore di opere creative; Deve includere il nome dell'autore, o un riferimento URL;
- `q`: inserisce contenuti citati da un'altra fonte;
- `var`: rappresenta una variabile;
- `samp`: rappresenta l'output di un programma o un sistema computer;
- `kbd`: rappresenta input dell'utente (da tastiera o da altre periferiche);
- `strong`: rappresenta forte importanza, serietà, o urgenza (cioè che il contenuto andrebbe visto per primo);
- `sup` o `sub`: apice o pedice;
- `i`: rappresenta un frammento di testo in una voce alternativa, o diverso dal testo che lo circonda, una frase idiomatica, un termine di un'altra lingua, ecc... (effettivamente scritto in corsivo);
- `b`: rappresenta un frammento di testo su cui si porta attenzione per motivi utilitari senza rappresentare maggiore importanza, come parole chiavi, nomi di prodotti, ecc... (effettivamente scritto in grassetto);
- `mark`: rappresenta un frammento di testo marchiato o evidenziato per motivi di riferimento, per via della sua importanza in questo o in un altro contesto.

3.2.4 Semantica di modifica

Esistono alcuni tag atti a specificare modifiche fatte al documento. Questi sono:

- `ins`: rappresenta un'aggiunta al documento;
- `del`: rappresenta una rimozione dal documento;

- `cite`: può essere usato per specificare l'indirizzo del documento che documenta la modifica;
- `datetime`: può essere usato per specificare la data e l'ora della modifica. Un esempio di questi tag può essere:

```
<h1>Audiofili e tendenze di mercato</h1>
<p>
  Le ultime tendenze mostrano che il <del>75%</del><ins>80%</><cite>
    studio</cite> degli utenti di apparecchi audio non sa distinguere
    la differenza fra l'mp3 e un <del>disco in vinile</del><ins>cd</
    ins><datetime>02/03/2024</datetime>
...
</p>
```

3.2.5 Embedding di contenuti

Diversi tag in HTML hanno lo scopo esplicito di includere contenuti multimediali. Questi sono:

- `img`: come già visto, una immagine può essere inclusa specificando `src` e `alt` (l'URL sorgente e un testo alternativo da visualizzare in caso di mancanza d'immagine).

  In un dato momento, un immagine può trovarsi in uno di 4 stati:

  - **Non disponibile:** l'immagine non è stata ricevuta, si visualizza `alt`;
  - **Parzialmente disponibile:** l'immagine è in fase di ricezione, si visualizza quanto ricevuto finora o `alt`;
  - **Completamente disponibile:** l'immagine è stata ricevuta, si hanno a disposizione almeno le dimensioni, quindi si visualizza;
  - **Danneggiata:** l'immagine non può essere ricevuta, oppure è stata ricevuta ma è corrotta / in un formato non supportato. Si visualizza `alt`.

  #### Image map

  Una image map specifica regioni dell'immagine che hanno funzioni specifiche (ottenere un documento, eseguire un programma, ecc...). Un'elemento `map`, collegato ad un elemento `img` e corredato dei figli `area` e un attributo `name` che permette di riferlo, forma un'image map.

  L'elemento `area` specifica una singola area all'interno dell'image map. Questo elemento ha gli attributi:

  - `alt`: specifica un testo alternativo per l'area;
  - `coords`: specifica le coordinate dell'area, secondo il tipo di area scelto;
  - `href`: specifica il link di destinazione di un area;
  - `download`: specifica se il link è usato per download;
  - `shape`: specifica la forma dell'area. A tipi di area diverse corrispondono formati di coordinate diversi, scegliendo fra:
    * `default`: solitamente `rect`;
    * `rect`: coordinate `left-x, top-y, right-x, bottom-y`;
    * `circle`: coordinate `center-x, center-y, radius`;
    * `poly`: coordinate `x1, y1, x2, y2, ..., xn, yn`.

  - `target`: specifica il target di apertura del link, come per `a`.

- `iframe`: rappresenta un contesto di navigazione innestato (cioè un file HTML dentro un file HTML). Il contenuto dell'iframe è definito secondo due modalità mutualmente esclusive:

  - `src`: si specifica un'attributo che contiene un URL alla risorsa interessata:

```
<iframe width="560" height="315"
    src="https://www.youtube.com/embed/SShGRVKI9xI?si=
    sx5QLk6ELUMsHwwU"
    ...
</iframe>
```

  - `srcdoc`: si specifica del codice html sul posto.

```
<iframe name="iframe" srcdoc="<p>HTML-ception</p>"></iframe>
<a href="/subpage.html" target="iframe">Link</a>
```

  Nell'ultimo esempio, si noti che il link ha come target l'iframe, ergo si aprirà dentro di esso. Si noti anche che nel caso di doppia definizione, `srcdoc` ha la precedenza. Su un iframe si hanno poi gli attributi:

  - `width` e `height`: determinano rispettivamente larghezza e altezza dell'iframe;
  - `sandbox`: abilita una serie di restrizioni sui contenuti gestiti dall'iframe (ne abbiamo viste alcune riguardo all'apertura di href in deterimati target dei tag `a`).

  Nel caso un iframe non sia visualizzabile, l'HTML compreso fra i tag dell'iframe viene visualizzato come fallback. Ad esempio:

```
<iframe src="pagina.html">
  <p>Il tuo browser non supporta gli iframe! Che peccato!</p>
  <a href="pagina.html">Link alla pagina</a>
</iframe>
```

  Infine, non si può avere nesting ricorsivo, ergo l'iframe non può essere un documento contenuto fra gli antenati del documento corrente.

- `embed`: rappresenta un contenuto esterno, tipicamente non-HTML, e interattivo. Anche qui, l'attributo `src` contiene l'URL della risorsa interessata. L'attributo `type`, invece, contiene il tipo MIME della risorsa, come specificato dalla IANA. Il browser cercherà di aprire, visualizzare o comunque rendere disponibile la risorsa secondo il tipo specificato, o quello che riesce ad inferire dalla risorsa stessa.

```
<embed src="filmato.mp4" width="320" height="240" title="Filmato
    Ganzissimo"/>
```

- `object`: rappresenta una risorsa esterna che, a seconda del tipo, verrà interpretata come un immagine, un contesto di navigazione innestato, o un'altro tipo di contenuto. Si può intendere come una versione più versatile dei tag visti finora.

  I suoi attributi sono:

  - `data`: specifica l'indirizzo della risorsa;
  - `type`: specifica il tipo MIME della risorsa;

  - `typemustmatch`: un booleano che indica se la risorsa va aperta solo se il suo tipo corrisponde a quello specificato.

  L'`object` supporta un meccanismo di fallback simile a quello degli iframe. Ad esempio:

```
<object data="documenti/statuto.pdf" type="application/pdf"
    typemustmatch="true">
  Non puoi visualizzare questo contenuto. Ecco un <a href="documenti/
    statuto.pdf">link</a>.
</object>
```

- `param`: rappresenta un parametro per i plugin invocati dagli elementi `object`, attraverso coppie `name` - `value`.

- `video`: rappresenta un video o un filmato, oppure file audio con sottotitoli, o ancora uno stream video. Ha gli attributi:

  - `src`: specifica l'URL della risorsa;
  - `autoplay`: indica che il video verrà riprodotto appena sarà stato caricato;
  - `controls`: specifica che i controlli dovrebbero essere mostrati (solitamente il tasto avvia, il controllo volume, la barra di seek, la durata del video, ecc...);
  - `width` e `height`: determinano rispettivamente larghezza e altezza del video;
  - `loop`: specifica che il video dovrebbe essere riprodotto da capo una volta terminato;
  - `muted`: specifica che l'audio dovrebbe essere mutato;
  - `poster`: specifica un'immagine da mostrare mentre il video viene caricato (se si è attivato anche `autoplay`), o finché l'utente non lo avvia manualmente;
  - `preload`: specifica come il video dovrebbe essere precaricato secondo tre modalità:
    * `auto`: il browser dovrebbe scegliere automaticamente;
    * `metadata`: il browser dovrebbe precaricare solo i metadati;
    * `none`: il browser non dovrebbe precaricare il video.

    Senza questo tag, si assume che sia impostato ad `auto`.

  Un'esempio dell'uso di questo tag è:

```
<video width="320" height="240" src="advert.mp4" autoplay poster="
    advert_thumb.jpg"/>
```

- `source`: rappresenta una sorgente alternativa per elementi multimediali. Ha gli attributi `src` e `type`. Ad esempio, si può usare per fornire più possibilità nel caso il browser non supporti il tipo di una risorsa:

```
<video width="320" height="240" controls>
  <source src="film.mp4" type="video/mp4">
  <source src="film.ogg" type="video/ogg">
  Il tuo browser non supporta questo contenuto.
</video>
```

- `audio`: rappresenta una risorsa o uno stream audio, come `video` rappresentava una risorsa o uno stream video:

```
<audio controls>
<source src="ambiance.mp3" type="audio/mpeg">
</audio>
```

- `link`: rappresenta un collegamento ad un altro file, in modo diverso da `a` (che starebbe per *anchor*): se `a` indicava un link cliccabile vero e proprio, `link` specifica una risorsa collegata al documento che va scaricata dal browser, come ad esempio gli stili CSS. Gli attributi sono:
  - `href`: l'URL della risorsa collegata;
  - `hreflang`: la lingua della risorsa collegata;
  - `media`: specifica una media query, cioè un'indicazione su per quali dispositivi è stata ottimizzata la risorsa;
  - `rel`: specifica la relazione fra il documento corrente e quello linkato, nelle modalità già viste per `a`. Anzi, si noterà che alcuni dei tipi `rel` hanno più significato per i `link` di quanto ne hanno per gli `a`;
  - `sizes`: dimensione delle icone nel caso sia impostato `rel="icon"`;
  - `type`: specifica il tipo MIME della risorsa collegata.

  Ad esempio, si avrà, per includere un file CSS:

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

4 Lezione del 07-10-24

4.1 CSS

I Cascading Style Sheets (CSS) sono uno standard per la descrizione della presentazione degli elementi HTML. In CSS, possiamo assegnare propietà come:

- Font;
- Colori;
- Dimensioni degli elementi;
- Posizioni degli elementi;
- Bordi;
- Immagini di sfondo;
- ...

In generale, il CSS è un linguaggio a sé stante. Si può aggiungere del CSS direttamente ad un elemento HTML attraverso l'attributo `style`, all'interno dell'intestazione `<head>`, o, più comunemente, in un file separato che contiene solo codice CSS.

4.1.1 Vantaggi del CSS

Ci sono diversi vantaggi all'uso del CSS:

- Il grado di controllo della formattazione in CSS è significativamente migliore di quello fornito da HTML;
- I siti web diventano più facili da mantenere quando tutta la formattazione è essere inserita all'interno di uno, o una piccola quantità, di file CSS;
- Si ottiene più facilmente uno stile grafico consistente fra pagine;
- I siti basati sul CSS sono più accessibili, in quanto l'HTML non deve preoccuparsi della presentazione grafica;
- Un sito creato con del CSS serializzato sarà anche più facile da scaricare in quanto ogni pagina conterrà meno codice (basta scaricare il CSS una volta sola);
- Col CSS si può adattare una pagina a più formati;

Detto questo, ci sono anche varie difficoltà associate al fatto che il CSS non è stato pensato come una lingua per il layout, ergo è complicato gestire elementi fluttuanti, posizioni relative, gestione delle altezze e dei margini, ecc...

4.1.2 Versioni del CSS

Il comitato W3C ha pubblicato la raccomandazione di CSS livello 1 nel 1996. Un'anno dopo è arrivata la raccomandazione di livello 2, nota semplicemente come CSS2. Una versione aggiornata della raccomandazione di livello 2, la CSS2.1, fu pubblicata dopo circa dieci anni di lavoro, nel 2011. Nel frattempo, la W3C stava lavorando anche ad una versione completamente diversa, la CSS3.

Alcune funzionalità della CSS3 sono entrate a far parte della raccomandazione W3C:

- Selettori;
- Namespace;
- Media query;
- Colori;
- Attributi di stile.

4.1.3 Sintassi del CSS

Un documento CSS consiste di una o più **regole** di stile. Una regola consiste in un selettore che identifica l'elemento o lgi elementi HTML interessati, seguita da una serie di dichiarazioni di coppie proprietà - valore. Ad esempio:

```
selector {
  property: value;
  property2: value;
}
```

Ogni lista di dichiarazioni si chiama anche **blocco** di dichiarazioni. Lo whitespace è ignorato, l'unica cosa importante sono i punti e virgola.

4.1.4 Commenti in CSS

Si possono inserire commenti nella forma classica: `/* Questo e' un commento */`.

4.2 Selettori

Quando si definiscono regole CSS, dobbiamo prima usare un selettore per indicare quali elementi verranno effettati. I selettori CSS possono riferirsi a:

- Elementi singoli;
- Elementi multipli;
- Elementi accoppiati in qualche modo;
- Elementi che sono posizionati in un modo specifico nella gerarchia.

4.2.1 Selettori di elemento

Usano il nome dell'elemento HTML. Esiste un selettore universale, indicato dal carattere asterisco.

4.2.2 Selettori raggruppati

Si possono raggruppare i selettori, usando le virgole:

```
p, div, aside {
  margin: 0;
  padding: 0;
}
```

Questo è un buon modo per ridurre le dimensioni dei file CSS (elementi con le solite proprietà possono essere ragguppati).

Spesso si usano i selettori raggruppati per annullare gli stili di default del browser, in modo da ridurre inconsistenze date dalla visualizzazione su software diversi.

4.2.3 Selettori di classe

Esiste un'attributo HTML, il `class`, che permette di specificare classi per elementi di tipo uguale o diverso. Tutti gli elementi di una solita classe possono essere selezionati in CSS, usando semplicemente un punto seguito dal nome della classe.

Poniamo di avere dell'HTML:

```
<h1 class="first">Primo</h1>
<p class="first">Sottotitolo</p>
...
```

Potremo stilizzare sia l'header che il paragrafo con il CSS:

```
.first {
  font-style: italic;
  color: red;
}
```

4.2.4 Selettori di ID

Un selettore di id permette di indicare un'elemento specifico, attraverso l'attributo `id`, usando un cancelletto seguito dall'id dell'elemento:

Poniamo di avere dell'HTML:

```
<h1 id="second">Primo</h1>
```

Potremo stilizzare l'header con il CSS:

```
#second {
  font-style: italic;
  color: red;
}
```

La differenza fra i selettori di classe e i selettori di id è che i secondi dovrebbero essere usati solo per riferirsi ad un **unico** elemento, mentre i primi vanno bene per un numero qualsiasi di elementi. Inoltre, come vedremo in seguito, la specificità dei selettori di id è maggiore di quella dei selettori di classe.

4.2.5 Selettori di attributo

Si possono selezionare elementi in base alla presenza o meno di determinati attributi all'interno di elementi, o di valori specifici dati ad attributi di determinati elementi. Per questo si usano le parentesi quadre:

```
[title] {
  cursor: help;
}
```

Questo codice significa che ogni elemento con l'attributo title mostrerà un cursore col simbolo del punto interrogativo (o qualsiasi sia lo stile definito dal sistema operativo per `help`).

Esistono, in verità, modi più sofisticati di selezionare su attributi:

- `[attribute]`: cerca elementi con un attributo specifico;
- `[attribute=value]`: cerca elementi con un attributo specifico impostato ad un certo valore;
- `attribute~=value1, value2`: cerca elementi con un attributo specifico impostato ad almeno uno dei valori definiti in una lista separata da virgole;
- `attribute^=value`: cerca elementi con un attributo specifico il cui valore inizia col valore specificato;
- `attribute*=value`: cerca elementi con un attributo specifico il cui valore contiene la stringa specificata;
- `attribute$=value`: cerca elementi con un attributo specifico il cui valore finisce col valore specificato.

4.2.6 Pseudo-selettori

Un selettore di pseudo-elemento è un modo di selezionare qualcosa che non esiste esplicitamente come un elemento nel documento HTML ma che rappresenta comunque un oggetto selezionabile. I selettori di pseudo-elemento cominciano con una coppia di due punti, anche se a volte va bene un singolo due punti.

Un selettore di pseudo-classe si applica ad un elemento HTML esistente, ma lo indirizza quando si trova in un certo stato, o in una certa relazione familiare.

Alcuni esempi sono:

- **Pseudo-classe:**
  - `a:link`: si applica a link non visitati;
  - `a:visited`: si applica a link visitati;
  - `:focus`: si applica all'elemento che attualmente ha focus;
  - `:hover` si applica all'elemento su cui sto facendo hover;
  - `:active`: si applica all'elemento attivo (e.g. un bottone cliccato);
  - `:checked`: si applica a elementi attivati, come i radio button;
  - `:first-child`: si applica al primo figlio di un'elemento.
- **Pseudo-elemento:**
  - `::first-letter`: si applica alla prima lettera di un elemento;
  - `::first-line`: si applica alla prima linea di un elemento;
  - `::before`: inserisce contenuti prima di un elemento;
  - `::after`: inserisce contenuti dopo un elemento.

4.2.7 Selettori contestuali

I selettori contestuali sono selettori di pseudo-classe che permettono di selezionare elementi basandosi sui suoi antenati, discendenti, o fratelli, ergo sulla base della loro relazione gerarchica con altri elementi nell'albero del documento.

Questi sono:

- **Discendente:** indica un elemento che è contenuto da qualche parte dentro un altro elemento. Ad esempio: `div p` seleziona un paragrafo dentro una div;
- **Figlio:** indica un'elemento specifico che è figlio diretto dell'elemento specificato. Ad esempio: `div>h2` seleziona un header (di peso 2) figlio diretto di una div;
- **Fratello adiacente:** indica un elemento che è il prossimo fratello (direttamente dopo) dell'elemento specificato. Ad esempio: `h3+p` seleziona il primo paragrafo dopo qualsiasi heading (di peso 3).
- **Fratello generale:** indica un qualsiasi elemento che condivide il genitore con l'elemento specificato. Ad esempio: `h3~p`: indica un qualsiasi paragrafo che condivide il genitore con un heading (di peso 3).

Un caso particolare è rappresentato dai selettori di tipo discendente, in quanto sono i più usati. Questi selezionano elementi contenuti da altri elementi, e si specificano usando il carattere spazio. Possono anche includere ulteriori informazioni sul tipo di discendenza dell'elemento, come ad esempio l'essere primi figli:

```
1 #main div p:first-child { ... }
```

4.3 Proprietà

Ogni dichiarazione in CSS deve contenere una proprietà (ce ne sono centinaia). I nomi delle proprietà sono definiti dallo standard.

Vediamone alcuni esempi:

- **Font:**
  - `font`;
  - `font-family`;
  - `font-size`;
  - `font-style`;
  - `font-weight`;
  - `@font-face`.
- **Testo:**
  - `letter-spacing`;
  - `line-height`;
  - `text-align`;
  - `text-decoration`;
  - `text-indent`.
- **Colori e sfondo:**
  - `background`;
  - `background-color`;
  - `background-image`;
  - `background-position`;
  - `background-repeat`;
  - `box-shadow`;
  - `color`;
  - `opacity`.
- **Bordi:**
  - `border`;
  - `border-color`;
  - `border-width`;
  - `border-style`;
  - `border-top`, `border-left`, `border-right`, `border-bottom`;
  - `border-image`;
  - `border-radius`.
- **Spaziature:**
  - `padding`;

  - `padding-bottom`, `padding-left`, `padding-right`, `padding-bottom`;
  - `margin`;
  - `margin-bottom`, `margin-left`, `margin-right`, `margin-bottom`;

- **Dimensioni:**
  - `height`;
  - `max-height`;
  - `min-height`;
  - `width`;
  - `max-width`;
  - `min-width`;

- **Layout:**
  - `bottom`, `left`, `right`, `top`;
  - `clear`;
  - `display`;
  - `float`;
  - `overflow`;
  - `position`;
  - `visibility`;
  - `z-index`.

- **Liste:**
  - `list-style`;
  - `list-style-image`;
  - `list-style-type`.

- **Effects:**
  - `animation`;
  - `filter`;
  - `perspective`;
  - `transform`;
  - `transition`.

4.4 Valori

Ogni dichiarazione deve contenere poi un valore per ogni properietà. Alcuni valori di proprietà vengono da una lista predefinita di parole chiave. Altri sono valori come lunghezze, percentuali, numeri puri, colori od URL.

Vediamoli nel dettaglio:

4.4.1 Colori

Esistono più modi di esprimere il colore in CSS:

- **Nome:** riferendosi per nome ai colori cercati, ad esempio `red`, `lightblue`, `hotpink` (solo CSS3), ecc...
- **RGB:** si puo indicare il colore in modalità RGB (Rosso, Verde e Blu) come `rgb(255,255,255)`. Inoltre, si può usare un quarto attributo *alpha*, che rappresenta la trasparenza del colore, in forma `rgb(255,255,255,0)`.
- **HSL:** si può indicare il colore anche in modalità HSL (Hue, Saturation, Lightness (sarebbe Value)) come `hsl(255,255,255)`. Anche qui è supportato l'attributo *alpha*.
- **Esadecimale:** infine, si possono indicare i colori in modalità RGB con caratteri esadecimali, ad esempio `#FFFFFF`.

4.4.2 Unità di misura

Si possono usare **unità relative**, cioè basate sul valore di qualcos'altro, come la dimensione di un genitore; e **unità assolute**, cioè basate su unità di misura reali.

Esempi di unità relative sono:

- `em`: uguali al valore della proprietà `font-size` dell'elemento su cui è usato. Quindi, quando si usa per le dimensioni dei font, l'unità `em` dipende dalle dimensioni del font del genitore;
- `%`: una misura relativa, in percentuale, alle dimensioni dell'elemento su cui viene usata;
- `ex`: simile all'em, ma si basa sull'altezza x del font;
- `ch`: ancora simile all'em, ma si basa sulla dimensione del carattere 0;
- `rem`: sta per *root em*, che è la dimensione di font dell'elemento radice; A differenza di `em`, che potrebbe variare fra elementi, `rem` è assoluto nella pagina;
- `vw`, `vh`: stanno per *viewport width* e *viewport height*, cioè le dimensioni della finestra dove viene visualizzata la pagina. Sono entrambe misure percentuali, e possono essere usate per creare elementi che scalano insieme alla pagina.

Ed esempi di unità assolute sono:

- `in`: pollici;
- `cm`: centimetri;
- `mm`: millimetri;
- `pt`: points, è la misura predefinita di `font-size`, e vale $\frac{1}{72}$ pollici;
- `pc`: pica, vale $\frac{1}{6}$ pollici;
- `px`: pixel, vale $\frac{1}{96}$ pollici.

4.5 Tipi di stylesheet

Uno stylesheet è un file CSS che viene applicato ad una pagina web. Esistono più tipi di stylesheet:

- **Stylesheet d'autore:** creati da zero per un certo sito o una certa pagina;
- **Stylesheet utente:** creati dagli utenti (o dagli sviluppatori dei browser), solitamente per motivi di accessibilità;
- **Stylesheet del browser:** forniti insieme al browser per visualizzare pagine senza CSS.

Inoltre, possiamo (come già visto) mettere il CSS in più posizioni all'interno del codice. Vediamole nel dettaglio:

- **Stili inline:** sono regole di stile inserite direttamente dentro un'elemento HTML attraverso l'attributo style:

```
<h2 style="font-size: 24pt">Questo e' un grande titolo</h2>
<h2 style="font-size: 24pt; font-weight: bold">Questo e' un
    grandissimo titolo</h2>
```

  Solitamente, conviene evitare di usare questo attributo, in quanto esistono modi migliori e più mantenibili per stilizzare gli elementi;

- **Stili embedded:** sono regole di stili incluse nell'elemento `style`, all'interno dell'head.

```
<head lang="en">
<title>La mia pagina</title>
  ...
<style>
h2 {
  font-size: 24pt;
  font-weight
}
</style>
</head>
```

  Rappresentano un'approccio migliore degli stili inline, ma sono comunque poco raccomandati in confronto all'opzione successiva;

- **Stili esterni:** sono file CSS esterni inclusi come riferimento nel documento HTML attraverso l'elemento `link`. Rappresentano l'opzione più mantenibile (lo stesso stylesheet può stilizzare più pagine, ergo aggiornare lo stylesheet significa aggiornare tutte le pagine), e più veloce (il browser deve fare una sola richiesta HTTP dello stylesheet per tutte le pagine del sito).

4.6 La cascata

CSS sta per *Cascading* Style Sheet. Cascading, in questo contesto, si riferisce al modo in cui si discriminano le regole da scegliere in caso di conflitti. Questo è necessario in quanto, come abbiamo appena visto, esistono più tipi di stylesheet che spesso si vanno a sovrapporre.

CSS usa i seguenti principi per la sua cascata:

- **Ereditarietà**;
- **Specificità**;
- **Posizione**.

Vediamoli nel dettaglio.

4.6.1 Ereditarietà

Molte (ma non tutte) le proprietà CSS si riferiscono non solo a loro stesse, ma anche ai loro discenedenti. Font, colori, properità di lista e testo sono ereditabili. Layout, dimensioni, bordi, sfondi e spaziature non lo sono di default, ma lo possono diventare attraverso la parola chiave `inherit` usata per specificare il valore della properità in un elemento discendente.

Ad esempio, potremo avere:

```
div {
  font-weight: bold;
  margin: 50px;
  border: 1pt solid green;
}
p {
  border: inherit;
  margin: inherit;
}
```

In questo caso, un `p` dentro un `div` avrà bordo e margini ereditati da quest'ultimo.

4.6.2 Specificità

La specificità determina quali regole hanno precedenza se applicabili allo stesso elemento. Più un selettore è specifico, più alta è la precedenza della regola che lo usa.

Questo è implementato con un sistema di **pesi**, dove diversi selettori hanno diversi pesi (ad esempio un selettore di id è più specifico di un selettore di classe, che a sua volta è più specifico di un selettore di elemento, ecc...) sulla specificità della regola.

La specificità viene calcolata come un numero concatenato $abcd$, dove a, b, c e d sono cifre. Le cifre si assegnano come segue:

- a: vale 1 se il CSS è inline a dell'HTML, 0 altrimenti;
- b: conta il numero di attributi id del selettore;
- c: conta il numero di altri attributi e pseudo-classi del selettore;
- d: conta il numero di elementi e pseudo-elementi del selettore.

4.6.3 Posizione

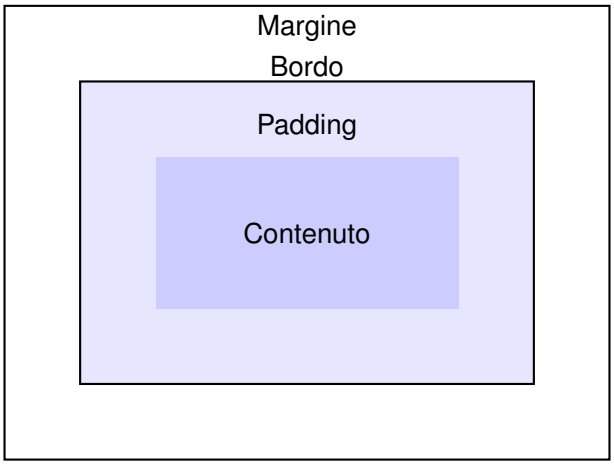
Esiste infine un'altro aspetto da considerare, cioè quello della posizione della regola. Il principio della posizione è che quando le regole hanno la stessa specificità, le ultime hanno più peso. Questo vale sia per le regole, sia per i blocchi di dichiarazioni.

Esiste un'eccezione a questa regola: il marchio `!important` fara in modo che la regola a cui è applicato sovrascriverà ogni altra regola definita dall'autore in altre posizioni.

4.7 Modello a scatola

Veniamo quindi a quello che è effettivamente il meccanismo secondo cui vengono disposti gli elementi sulla pagina. In HTML, tutti gli elementi esistono dentro una **scatola**. La scatola ha `width` e `height`, cioè larghezza e altezza, un `padding`, cioè dell'area in più *interna* alla scatola, un `margin`, cioè un'area di margine *esterna* alla scatola, è un bordo detto `border`.

Il contenuto dell'elemento stà all'interno della scatola, fino al padding. Lo sfondo del padding è il `background-color` o la `background-image` dell'elemento, mentre lo sfondo del margine è il `background-color` o la `background-image` del genitore dell'elemento.



4.7.1 Sfondi

Oggi gli sfondi degli elementi, o *background*, vengono realizzati con apposite proprietà CSS, e non con il tag `<img>`.

Vediamo quali sono queste proprietà:

- `background`: una proprietà combinata che permette di impostare più valori del background contestualmente, imposta comunque a valori di default gli attributi non specificati;
- `background-attachement`: specifica se l'immagine di sfondo scorre col documento (comportamento di default) o rimane fissa. I valori possibili sono quindi `fixed` o `scroll`;
- `background-color`: specifica il colore dello sfondo;
- `background-image`: specifica l'immagine di sfondo attraverso un URL;
- `background-position`: specifica dove l'immagine dovrebbe essere posta: ha possibili valori `top`, `bottom`, `center`, `left` e `right`. Si possono anche fornire posizioni in pixel o in percentuali, sempre per entrambe le dimensioni, che indicano la distanza dall'angolo in alto a sinistra dell'elemento;
- `background-repeat`: determina la ripetizione del background, che può essere `repeat`, `repeat-x`, `repeat-y` e `no-repeat`.
- `background-size`: specifica la dimensione dell'immagine di sfondo.

4.7.2 Bordi

I bordi possono vengono usati per separare gli elementi fra di loro. Hanno le seguenti proprietà:

- `border`: una proprietà combinata che permette di impostare più valori del bordo contestualmente, imposta comunque a valori di default gli attributi non specificati;
- `border-style`: specfica il tipo di bordo. Sono supportati, fra l'altro, `solid`, `dotted`, `dashed`, `double`, `groove`, `ridge`, `inset` e `outset`.
- `border-width`: specifica la larghezza del ordo in unità (ma non in percentuali). Sono supportate anche le parole chiave `thin`, `medium`, ecc...
- `border-color`: specifica il colore del bordo;
- `border-radius`: specifica il raggio delle smussature degli spigoli;
- `border-image`: specifica l'URL di un'immagine da usare come bordo.

Si possono indicare questi attributi su un solo lato del bordo usando notazioni come `border-top-color`. Esiste anche una nozione più veloce, che permette di impostare le properietà di ogni lato con una sola dichiarazione, assumendo l'ordinamento `top`, `right`, `bottom` e `left` per i lati. Quindi, ad esempio, si potrà avere:

```
border-color: red green orange blue;
/* equivale a: */
border-top-color: red;
border-right-color: green;
border-bottom-color: orange;
border-left-color: blue;
```

Infine, si ricorda che i bordi vengono allargati dal padding, ma non dai margini.

4.7.3 Margini e padding

I margini indicano la dimensione dello spazio lasciato **esternamente** alla scatola, mentre il padding indica la dimensione lasciata **internamente** alla scatola. Si nota che i margini possono **collassare**: quando abbiamo margini adiacenti verticali, viene usato il valore del margine più grande, e l'altro viene annullato. I margini orizzontali invece non collassano mai, ed esistono alcuni casi particolari quando nemmeno i verticali lo fanno (il piacere di scoprirli è lasciato al lettore).

4.7.4 Dimensioni

Avevamo le proprietà `width` e `height` per impostare le dimensioni. Queste non sono del tutto veritiere, in quanto la dimensione degli elementi dipende anche dai loro contenuti. Per gestire questa situazione, si hanno a disposizione la proprietà `overflow`, che indica come gestire la dimensione in eccesso:

- `visible`: lascia i contenuti in eccesso visibili:
- `hidden`: nasconde i contenuti in eccesso (si comporta quasi come una maschera);
- `scroll`: visualizza scrollbar per permettere di visualizzare tutti i contenuti nel fattore forma indicato da `width` e `height`;
- `auto`: lascia scegliere al browser (spesso creera una scroll verticale).

4.8 Stilizzazione del testo

Esistono due tipi di proprietà che influenzano il testo:

- **Proprietà dei font** che influenzano i font e il loro aspetto;
- **Proprietà di paragrafo** che influenzano il testo a discapito del font usato.

Bisogna notare che i font usati dal browser devono essere installati sul computer dell'utente: questo significa che i font installati sulla macchina dello sviluppatore potrebbero non trovarsi su quella degli utenti, e dare così problemi di compatibilità. Per questo motivo, solitamente si fornisce uno **web font stack**, cioè una lista di font alternativi da usare nel caso non si trovasse la scelta originale sul computer dell'utente.

Vediamo quindi tutte le proprietà dei font:

- `font`: una proprietà combinata che permette di impostare più valori del bordo contestualmente, imposta comunque a valori di default gli attributi non specificati;
- `font-family`: specifica quale font usare. Come già detto se ne possono (e se ne dovrebbero) specificare più di uno;
- `font-size`: specifica la dimensione del font in punti; `font-style`: specifica se il font è in corsivo, grassetto, ecc...
- `font-variant`: specifica la variante del font, ad esempio se è in stampatello minuscolo o no;
- `font-weight`: specifica il peso del font, con un valore da 100 a 900 in multipli di 100, o con una parola chiave fra `normal`, `bold`, `bolder` e `lighter`.

e del paragrafo:

- `letter-spacing`: specifica la spaziatura fra le lettere;
- `line-height`: specifica l'altezza di una linea di testo;
- `text-align`: specifica l'allineamento del testo;
- `text-decoration`: specifica le decorazioni (sottolineatura, sopralineatura, ecc...);
- `text-direction`: specifica la direzione del testo;
- `text-shadow`: imposta un'effetto di ombreggiatura, specificando gli offset orizzontali e verticali, la sfocatura e il colore dell'ombra.

4.8.1 Font-face

Il selettore `@font-face` permette di usare font non scaricati sulla macchina dell'utente, appoggiandosi a siti open source come Google Web Fonts.

4.8.2 Nota sulle unità di misura

Usare i pixel non è particolarmente sicuro: la densità dei pixel dei dispositivi moderni è variata in modo che un pixel CSS non corrisponde più tanto spesso a un pixel effettivo sul dispositivo. Se vogliamo creare layout che funzionino bene su più dispositivi, dobbiamo usare unità come gli `em` e le percentuali. Usare queste unità implica anche che l'utente potrà, a piacimento, cambiare la dimensione del testo. Ricordiamo poi l'unità `rem`, che si basa sull'origine del documento, ed è quindi sempre uguale in esso.

4.9 Tabelle HTML

Il linguaggio HTML ci permette di creare tabelle attraverso il tag `table`. Queste possono visualizzare una vasta gamma di contenuti, dalle immagini al testo, ai link e altre tabelle.

4.9.1 Struttura di una tabella

Una `table` contiene un certo numero di righe indicate da `tr`, e ogni riga contiene un certo numero di celle `td`. I contenuti vanno sempre dentro le celle. I titoli delle colonne si scrivono nei tag `th` anziché `td`. Ad esempio, potremmo avere:

```
<table>
  <tr>
    <th>Sintetizzatore</th>
    <th>Produttore</th>
    <th>Anno</th>
  </tr>
  <tr>
    <td>Roland TB-303</td>
    <td>Roland</td>
    <td>1981</td>
  </tr>
  <tr>
    <td>Yamaha DX7</td>
    <td>Yamaha</td>
    <td>1983</td>
  </tr>
  <tr>
    <td>ARP Odyssey</td>
    <td>ARP</td>
    <td>1972</td>
  </tr>
  <tr>
    <td>Sequential Prophet-5</td>
    <td>Sequential Circuits</td>
    <td>1978</td>
  </tr>
</table>
```

4.9.2 Celle non standard

Se vogliamo che una cella occupi più colonne, usiamo l'attributo `colspan`. Ad esempio:

```
<table>
  <tr>
    <th>Sintetizzatore</th>
    <th colspan=2>Anno e produttore</th>
  </tr>
  ...
</table>
```

Allo stesso modo, se vogliamo che una cella occupi più righe, usiamo `rowspan`. Ad esempio:

```
<table>
  <tr>
    <th>Produttore</th>
    <th>Sintetizzatore</th>
  </tr>
  <tr>
```

```
    <td rowspan=2>Roland</td>
    <td>MS-20</td>
  </tr>
  <tr>
    /* qui serve un'elemento solo! */
    <td>M1</td>
  </tr>
  <tr>
    <td>ARP Odyssey</td>
    <td>ARP</td>
    <td>1972</td>
  </tr>
  <tr>
    <td>Sequential Prophet-5</td>
    <td>Sequential Circuits</td>
    <td>1978</td>
  </tr>
</table>
```

4.9.3 Elementi addizionali

Esistono altri elementi da aggiungere alle tabelle:

- `caption`: indica un titolo per la tabella;
- `col` e `colgroup`: indicano gruppi di colonne;
- `thead`, `tbody` e`tfoot`: indicano una separazione semantica fra l'intestazione, il corpo e il footer di una tabella.

Le tabelle, in ogni caso, possono poi essere stilizzate come tutti gli elementi HTML attraverso il css.

5 Lezione del 14-10-24

5.1 Form

I form in HTML forniscono all'utente un modo di fornire dati al web server, ergo di digitare testo, password, impostare opzioni, ecc... In un ciclo di esecuzione standard, l'utente invia una richiesta HTTP al server, richiedendo un comune documento HTML. Il server risponde con un documento che contiene un form, che viene compilato dall'utente, e che viene quindi restituito al server attraverso attraverso una richiesta GET o POST. I dati ottenuti dal server vengono quindi elaborati attraverso script in PHP o altre tecnologie.

Si noti che l'elaborazione dei dati del form può essere effettuata anche dal browser attraverso, ad esempio, codice JavaScript.

La dichiarazione di form si fa attraverso l'elemento `form`. Questo elemento comprende di due attributi essenziali:

- `action`: specifica l'URL della risora lato server che si occuperà dell'elaborazione del form (ad esempio uno script in PHP);
- `method`: specifica il modo in cui la stringa di query verrà trasmessa dal browser al server. La **stringa di query** rappresenta le informazioni ottenute dal form sotto forma di una lista di coppie chiavi-valore in forma `key=value` separata da `&`. Ad esempio, una possibile stringa di query dal form di registrazione di un utente potrebbe essere: `nome=pippo&cognome=pippo&codicefiscale=pppppp...`.

Esistono due metodi di trasmissione delle stringhe di query:

- **GET**: si appende la stringa di query, preceduta da un `?`, all'URL della risorsa ottenuta dal sito. In questo modo, la richiesta è visible nella barra di ricerca del browser, e i dati vengono immagazzinati nella cache (non solo del browser, ma anche dei proxy). Inoltre, si ha un limite, solitamente di 2048 caratteri, sulla lunghezza della stringa. Bisogna quindi usare questo metodo nel caso si stiano richiedendo dati dal server o comunque si stiano trasmettendo dati non sensibili;
- **POST**: la stringa è inviata sotto forma di una richiesta POST, che è quindi nascosta dall'utente, solitamente non immagazzinata in cache, e può contenere dati in formato binario. Si usa quindi questo metodo nel caso si debbano inviare informazioni sensibili come password, ecc...

5.1.1 Elementi di controllo form

Esistono diversi elementi HTML per il controllo dei form:

- `button`: definisce un pulsante cliccabile dall'utente, con l'opzione di definire un'azione attraverso l'attributo `type`. Un pulsante può come vedremo, anche chiamare una funzione JavaSript attraverso l'attributo `onclick`.

  Esistono, nello specifico, due modi principali di definire un pulsante:

  - `input`: l'elemento `input` diventa un pulsante quando si usa l'attributo `type` con valore `submit`, `reset` o `button`. Nel primo caso si ha un pulsante per l'invio del form (attraverso la modalità stabilita in `method`), nel secondo un pulsante per il reset dei dati inseriti finora nel form, e infine nel terzo un pulsante qualsiasi con del testo definito dall'attributo `value`. Si può anche definire un'immagine cliccabile attraverso il tipo `image`. In questo caso si specifica l'immagine con l'attributo `src`. Esempi di questi casi sono:

```
/* un pulsante per il submit */
<input type="submit" />
/* un pulsante per il reset */
<input type="reset" />
/* un pulsante qualsiasi */
<input type="button" value="Cliccami!"/>
/* un immagine cliccabile */
<input type="image" src="images/icons/calendar.png"/>
```

  - `button`: l'elemento `button` è specifico ai pulsanti, e permette più customizzazione (si possono usare tag bold, enfasi, ecc...). In questo caso, il tipo del pulsante (cioè l'azione compiuta) si indica sempre con l'attributo `type`. Un esempio è:

```
<button type="submit">
<img src="images/icons/edit.png" alt="Modifica"/>
</button>
```

- `datalist`: definisce una lista di elementi da essere usata con altri elementi di controllo form, indicandola attraverso l'apposito attributo `id`. Ad esempio:

```
<datalist id="tipo-prodotto">
  <option value="Amplificazione e PA">
  <option value="Effetti">
  <option value="Luci e proiettori">
```

```
  <option value="Macchine del fumo">
</datalist>
```

- `fieldset`: raggruppa più elementi in un form, solitamente accompagnato da un'elemento `legend`.
- `input`: definisce un elemento di input. Molti altri elementi si possono ridurre ad elementi di input, tra cui, come abbiamo visto, i pulsanti. Nello specifico possiamo individuare, riguardo al testo:
  - `text`: una casella di testo su una sola linea per input generico;
  - `password`: una casella di testo su una sola linea per password, che occulta i caratteri inseriti dall'utente;
  - `search`: una casella di testo su una sola linea per query di ricerca (ad esempio la barra di inserimento di un motore di ricerca);
  - `email`: una casella per indirizzi di posta elettronica;
  - `tel`: una casella per numeri di telefono;
  - `url`: una casella per inserire URL.

  Sono disponibili poi attributi avanzati, tra cui il `pattern`, che permette di definire un'espressione regolare che l'ingresso testuale dell'utente dovrebbe rispettare, e il `list`, che permette di riferirsi ad una `datalist` definita precedentemente per l'autocompletamento del testo digitato. Per altri tipi di validazioni esistono poi i tipi `number` e `slider`, che supportano gli attributi `min`, `max` e `step` (perlopiù usati su input di tipo numerico):

```
<label>Eta'</label>
<input type="number" min="1" max="99" name="age"/>

<label>Quanto sei soddisfatto dei nostri servizi?</label>
<p>
Per niente
<input type="slider" min="0" max="100" step="1" name="satisf">
Molto
</p>
```

  Per quanto riguarda interruttori di vario tipo, invece, si ha:

  - `radio`: permette di definire radio button, cioè pulsanti a selezione mutualmente esclusiva, distinguendo attraverso l'attributo `name`:

```
<input type="radio" name="dove">A casa</input>
<input type="radio" name="dove">A lavoro</input>
<input type="radio" name="dove">A scuola</input>
```

    Supporta anche gli attributi `value`, che ha lo stesso funzionamento che ha in `option`, e `checked`, per specificare un valore di default.
  - `checkbox`: permette di definire checkbox per ottenere risposte si/no dagli utenti. Si possono raggruppare checkbox con l'attributo `name`, e raggrupparle con l'attributo `name`, come accadeva per i radio button:

```
<label>Accetto le condizioni</label>
<input type="checkbox" name="accept"/>
```

    Esistono poi tipi di input specializzati, fra cui notiamo:

    * `hidden`: una casella di input nascosta;
    * `file`: apre una finestra per la selezione di file dal filesystem della macchina dell'utente;
    * `color`: apre un selettore di colori in formato RGB o HSL;
    * `date`: esistono diversi tipi di selettori temporali fra cui:
        - `date`: una data semplice in formato `yyyy-mm-dd`;
        - `time`: un orario in formato `HH:MM:SS`;
        - `datetime`: una combinazione di data e orario comprendente di un fuso orario;
        - `datetime-local`: una combinazione di data e orario senza specifica del fuso orario;
        - `month`: un mese in formato `yyyy-mm`;
        - `week`: una settimana dell'anno in formato `yyyy-WW#`.
    * `progress` e `meter`: elementi che indicano il progresso di una certa operazione, vanno scriptati attraverso JavaScript;
    * `output`: contiene l'output di un calcolo (ad esempio di una calcolatrice);
    * `keygen`: contiene una chiave privata per la crittografia a chiavi pubbliche.

- `textarea`: definisce una casella di testo su più linee, il cui numero può essere specificato con l'attributo `rows`. Risulta quindi più efficace di un elemento `input` di tipo `text` per testo di dimensioni più grandi. Si può definire del testo predefinito fra i tag stessi della `textarea`, o attraverso l'attributo `placeholder`.

```
<textarea>
  Inserisci un po' di testo
</textarea>
/* equivale a: */
<textarea placeholder="Inserisci un po' di testo">
</textarea>
```

- `label`: definisce un'etichetta testuale per un'elemento di input. Solitamente si raggruppa una `label` con l'elemento corrispondente, all'interno di un paragrafo:

```
<p>
  <label>Nome</label>
  <input type="text"/>
</p>
<p>
<p>
  <label>Cognome</label>
  <input type="text"/>
</p>
  <label>Codice Fiscale</label>
  <input type="text" pattern="^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]
    $"/>
</p>
```

- `legend`: definisce un nome per un `fieldsest`.

- `option`: definisce un'opzione in una lista multi-item (come ad esempio un `select` o un `datalist`). Esiste anche `optgroup`, che definisce un insieme di opzioni correlate fra di loro in una lista multi-item sotto il nome `label`. Si può definire un valore attraverso l'attributo `value` da spedire al server in caso di invio del form, altrimenti si usa il testo contenuto fra i tag.

```
<select name="scelte">
  <optgroup label="Pizza">
    <option>Margherita</option>
    <option>Capricciosa</option>
  </optgroup>
  <optgroup label="Primi">
    <option>Pasta</option>
    <option>Polenta</option>
  </optgroup>
  ...
</select>
/* oppure */
<select name="scelte">
  <optgroup label="Pizza">
    <option value="pizza-margherita">Margherita</option>
    <option value="pizza-capricciosa">Capricciosa</option>
  </optgroup>
  <optgroup label="Primi">
    <option value="primi-pasta">Pasta</option>
    <option value="primi-polenta">Polenta</option>
  </optgroup>
  ...
</select>
```

- `select`: definisce un selettore multilinea per una lista multi-item. Gli elementi della lista vengono definiti attraverso i tag `option` e `optgroup`.

5.2 Layout in CSS

Vediamo adesso le funzionalità che il CSS mette a disposizione per modificare il layout, cioè cambiare le posizioni in cui gli elementi vengono disposti sulla pagina.

5.2.1 Controllo di flusso

Chiamiamo **flusso normale** il modo in cui il browser dispone gli elementi di norma. Possiamo distinguere fra due classi di elementi per quanto riguarda il flusso normale:

- **Elementi blocco:** sono contenuti, uno ad uno, su una nuova linea. Sono esempi di elementi blocco `p`, `div`, `h1 - h6`, `ul`, `ol`, `table`, ecc...
- **Elementi inline:** sono visualizzati sulla stessa linea se possibile, altrimenti sulla linea successiva (cioè tornano a capo come il testo). Sono esempi di elementi inline `em`, `a`, `img`, `span`, ecc...

Notiamo come `div` e `span` sono gli elementi di "default" che si dispongono, rispettivamente, a blocco e in linea, cioè che non hanno altra funzione se non di indicare la modalità di visualizzazione dei rispettivi figli.

Si può usare il CSS per modificare le modalità di visualizzazione degli elementi, attraverso la proprietà `display`. Ad esempio, per visualizzare uno span a blocco e un elemento di lista in linea, diremo:

```
span { display: block; }
li { display: inline; }
```

5.2.2 Posizionamento

Il CSS definisce la proprietà specifica `position` per il posizionamento di elementi rispetto al flusso normale. Una volta specificata la modalità di posizionamento, si indica la posizione vera e propria dell'elemento interessato attraverso le proprietà `top`, `bottom`, `left` e `right`, che indicano rispettivamente la posizione dall'alto e da sinistra. I valori ammessi sono:

- `absolute`: in posizionamento assoluto, un'oggetto viene rimosso completamente dal flusso normale. Ciò significa che il resto della pagina viene disposto secondo le regole del flusso normale, o secondo regole CSS, ma comunque ignorando l'elemento in questione. La posizione effettiva dell'elemento viene decisa relativamente (in modo poco intuitivo rispetto al nome) al primo antenato posizionato;
- `fixed`: in posizionamento fisso, un'elemento resta nella stessa posizione sulla pagina a prescindere dallo scorrimento verticale. Questa modalità e quindi usata per banner, intestazioni, o comunque elementi che devono essere sempre visibili;-
- `relative`: in posizionamento relativo, l'elemento viene spostato come in posizionamento assoluto, ma il flusso normale prosegue come se l'oggetto fosse sempre al suo posto. Questo significa che l'elemento può effettivamente essere spostato *rispetto* alla sua posizione iniziale senza alterare il resto della pagina;
- `static`: è il valore predefinito, cioè equivale al flusso normale.

Bisogna notare che i posizionamenti, sia assoluti che relativi, avvengono all'interno di un certo **contesto di posizionamento**, che abbiamo detto è il primo antenato posizionato. In caso non ci sia un antenato posizionato, il contesto di posizionamento è la pagina stessa.

5.2.3 Indice Z

Un attributo che si applica *soltanto* a elementi posizionati, e *soltanto* rispetto agli elementi su cui sono posizionati, è `z-index`, che indica l'indice di profondità di un elemento. Elementi con indice Z più alto vengono disposti sopra agli altri, e di contro elementi con indice Z più basso vengono disposti sotto gli altri.

5.2.4 Elementi fluttuanti

Un elemento fluttuante viene diposto a sinistra (`float: left`) o a destra (`float: right`) del flusso normale, che viene a disporsi di fianco all'elemento fluttuante. Questa modalità è utile per elementi che devono apparire all'interno del flusso del testo, come ad esempio immagini o tabelle. Si nota che un elemento fluttuante deve avere una larghezza specificata minore del 100%, in quanto altrimenti andrà a coprire di default l'intera larghezza della pagina (quindi rendendo inutile il fatto che sia fluttuante).

Più elementi fluttuanti cercano di disporsi sulla stessa linea. Nel caso un elemento debba essere il primo di una linea, si usa la proprietà `clear`, sempre con valori `left` o `right`, che impedisce a elementi fluttuanti di disporsi rispettivamente a sinistra o a destra dell'elemento indicato.

Nel caso un'elemento fluttuante sia contenuto in un contenitore, affinché questo non si riduca alle sole dimensioni del padding, si deve usare la proprietà `overflow` con attributo `auto`: questo obbligherà il contenitore ad espandersi fino a contenere qualsiasi elemento fluttuante sia suo figlio, qualunque sia la posizione in cui si vada a trovare.

5.2.5 Elementi nascosti

Si possono nascondere elementi sulla pagina attraverso due metodologie principali: attraverso la proprietà `display` con valore `none`, o attraverso la proprietà `visibility` con valore `hidden`. La prima, oltre a rendere invisibile l'elemento, lo rimuove dal flusso, e quindi modifica la disposizione del resto degli elementi sulla pagina. La seconda, invece, mantiene il flusso del resto della pagina invariato.

5.2.6 Flexbox

Per realizzare elementi affiancati in maniera più complessa di quanto reso possibile dal meccanismo degli elementi fluttuanti, si usa il Flexible Box Layout Model, o flexbox. Un contenitore in modalità di visualizzazione `display: flexbox` contiene diversi elementi affiancati, che vanno a disporsi diversamente sulla pagina a seconda delle dimensioni.

Un elemento figlio di un flexbox diventa automaticamente un flex item. Ogni flex item è fornito della proprietà `flex-basis`, di default `auto`, che indica la dimensione dell'item nel flex box.

La proprietà fondamentale del flexbox è `flex-direction`, cioè la direzione di in cui si vanno a disporre gli elementi. Questa ha valori:

- `row`: gli elementi si dispongono su una linea orizzontale. è l'impostazione di default;
- `row-reverse`: gli elementi si dispongono sempre su una linea orrizontale, ma al contrario;
- `column`: gli elementi si dispongono su una linea verticale;
- `column-reverse`: gli elementi si dispongono semper su una linea verticale, ma al contrario.

La direzione verso destra o sinistra (o verso l'alto o il basso) è definita dal tipo di layout del browser, solitamente LTR (Left To Right), ma che può essere RTL (Right To Left) per alcuni locale (ad esempio l'Arabo), o addirittura dall'alto verso il basso (ad esempio nel caso di alcuni font giapponesi).

Di norma, se si hanno troppi elementi per lo spazio disponibile, questi escono dai confini del flexbox. Si può ordinare agli elementi di disporsi su più righe nel caso di overflow attraverso la proprietà `flex-wrap: wrap`. Si possono combinare direzione e wrap in un unica proprietà, `flex-flow`. Ad esempio, si può avere `flex-flow: column wrap`.

Esiste poi la proprietà `flex`, applicata ai flex item, che determina come gli elementi stessi vanno a riempire lo spazio disponibile. Si nota che questo si applica di default a layout orizzontali, in quanto su layout verticali bisogna introdurre artificialmente spazio libero (di norma un contenitore si restringerebbe a contenere gli elementi nel minor spazio possibile). Si hanno quindi i valori per `flex`:

- `initial`: il valore di default di `flex` è `initial`, che si limita ad affiancare gli elementi da sinistra verso destra;
- `auto`: il valore `auto` cerca invece di allargare ogni elemento il più possibile in modo da riempire tutto lo spazio possibile.

  Implicitamente, questo valore imposta le proprietà del flex item:

  - `flex-grow`, che permette agli elementi di accrescere in dimensioni più della loro `flex-basis`, cioè appropiarsi di eventuale spazio libero fornito ad altri elementi;
  - `flex-shrink`, che permette agli elementi di rimpicciolire in dimensioni più della loro `flex-basis`

  In sostanza, la `flex-basis`, cioè la dimensione dei figli del flexbox, è ignorata, e questi vengono disposti in modo da riempire lo spazio.

La dimensione di ogni elemento può poi essere modificata attraverso la proprieta `flex`, su cui si può definire la dimensione assoluta, o una tripla di valori `flex-grow flex-shrink flex`.

5.2.7 Allineamento nei flexbox

Esistono due direzioni di allineamento nei flexbox: quella parallela alla direzione di disposizione degli elementi, e quella perpendicolare. Per la prima direzione, si usa `justify-content`, che può avere i valori:

- `flex-start`: dispone gli elementi all'inizio del flexbox;
- `flex-end`: dispone gli elementi a partire dal fondo del flexbox;
- `center`: mantiene le distanze fra gli elementi inalterate, e li centra semplicemente sull'asse verticale o orizzontale di scorrimento;
- `space-around`: allarga le distanze fra gli elementi in modo da riempire tutto lo spazio disponibile, mantenendo i margini;
- `space-between`: allarga le distanze fra gli elementi in modod da riempire tutto lo spazio disponibile, senza lasciare margini;
- `space-evenly`: come `space-around`, ma mantiene lo spazio fra i bordi e i singoli elementi il più uguale possibile.

L'allineamento sulla direzione perpendicolare si fa con `align-content`, che ha gli stessi valori visti finora, più il valore `stretch`, che estende gli elementi in dimensioni fino a riempire lo spazio perpendicolare disponibile.

Si ha poi che ogni singolo elemento può dichiarare il suo `align-self`, cioè la sua modalità personale di allineamento.

5.2.8 Griglie

Il modulo per le griglie del CSS permette di creare layout a griglia impostando diversi parametri. Innanzitutto, qualsiasi elemento in modalità di visualizzazione `grid` o `grid-inline` diventa automaticamente una griglia CSS.

A questo punto si possono specificare le proprietà di base:

- `grid-template-columns`: indica la larghezza delle colonne della griglia. Per questa misura è comodo usare l'unita `fr`, cioè frazionaria, che indica quale frazione dello spazio disponibile ogni colonna dovrebbe usare.

  CSS fornisce poi la funzione `repeat()` per creare layout a griglia con un numero arbitrario di colonne. Questa può assumere diverse forme:

  - `repeat(num-repeat, width)`: indica `num-repeat` colonne di larghezza `width`;
  - `repeat(num-repeat, width1, width2, ...)`: si possono specificare più larghezze, che anche in questo caso vengono ripetute, complessivamente, `num-repeat` volte;
  - `repeat(auto-fill, width)`: col parametro `auto-fill`, le colonne vengono ripetute fino a riempimento dello spazio orizzontale disponibile;
  - `repeat(auto-fill, minmax(min-width, max-width))`: con la funzione `minmax`, si può specificare una dimensione minima e una dimensione massima, rispettivamente di rimpicciolimento e di allargamento degli elementi nella griglia, che verranno quindi ridimensionati in modo da riempire lo spazio nella griglia.

- `grid-template-rows`: indica l'altezza delle righe della griglia. Per questa misura è comodo invece usare l'unità `px`, se non lasciarla completamente non specificata, in quanto la dimensione verticale della pagina dovrebbe essere decisa dai contenuti che ogni casella della griglia contiene. In ogni caso, è disponibile anche la funzione `repeat()` con tutti i possibili parametri visti prima;

- `gap`: indica i gap, cioè gli spazi, lasciati fra caselle della griglia. Si può usare la proprietà composita `gap`, che si aspetta due dimensioni, o le proprietà separate su colonna e riga `column-gap` e `row-gap`.

Esistono poi proprietà, relativa o alla griglia stessa o ai suoi elementi figli, che specificano la disposizione degli stessi all'interno della griglia. Per i singoli elementi, si hanno le proprietà:

- `grid-column-start`: indica la colonna di partenza dell'elemento;

- `grid-column-end`: indica l'ultima colonna dell'elemento: si noti che attraverso questa proprietà si possono creare elementi multi-colonna;

- `grid-row-start`: indica la prima riga dell'elemento;

- `grid-row-end`: indica l'ultima riga dell'elemento: come prima, si noti che attraverso questa proprietà si possono creare elementi multi-riga.

A livello riga, invece, si può usare la proprietà `grid-template-areas`, che permette di specificare la posizione di elementi figli, definita per ognuna di essi una specifica `grid-area`, attraverso una sintassi intuitiva direttamente nel codice:

```
body {
  display: grid;
  grid-template-areas:
    "head head head"
    "navi main tang"
    "mess foot tang";

  grid-template-columns: 1fr 3fr 1fr;
  grid-template-rows: 80px 1fr 100px;
  width: 100%;
  height: 1000px;
}

header {
  grid-area: head;
}
```

```
17
18 nav {
19   grid-area: navi;
20 }
21
22 main {
23   grid-area: main;
24 }
25
26 aside {
27   grid-area: tang;
28 }
29
30 .mess {
31   grid-area: mess;
32 }
33
34 footer {
35   grid-area: foot;
36 }
```

Infine, il singolo elemento può allinearsi all'interno della casella (o caselle) allocatagli nella griglia attraverso le proprietà `align-self` (riguardo all'allineamento verticale) e `justify-self` (allineamento orizzontale). Queste supportano i valori:

- `stretch`: l'elemento riempie in dimensioni la casella corrispondente;
- `start`: l'elemento si dispone all'inizio della casella corrispondente;
- `center`: l'elemento si dispone centralmente alla casella corrispondente;
- `end`: l'elemento si dispone in fondo alla casella corrispondente.

Queste proprietà possono essere modificate per tutti gli elementi figli dall'elemento griglia, attraverso le proprietà `align-items` e `justify-items`.

5.2.9 Multicolonna

CSS permette di realizzare layout multicolonna, cioè a scorrimento del testo su più colonne come quelle che potremmo trovare in un giornale. Le proprietà disponibili sono:

- `column-count`: indica il numero di colonne. Di default vale 1, ergo basta impostare questo valore per rendere un'elemento multicolonna;
- `column-gap`: indica lo spazio lasciato fra le colonne;
- `column-rule-style`, `column-rule-width` e `column-rule-width`: indicano lo stile, come si farebbe per un `border`, di una linea separatrice fra le colonne. Esiste anche la proprietà composita `column-rule`;
- `column-width`: indica una larghezza (solo suggerita, il browser formatterà come crede necessario il multicolonna) ideale per le colonne. Esiste il composito `column` per specificare `column-count` e `column-width` contestualmente;

Ogni elemento può poi disporsi su più colonne attraverso la proprietà `column-span`, che indica appunto su quante colonne dovrebbe disporsi l'elemento.

5.3 Media query e responsive design

Uno dei problemi più grandi dello sviluppo web moderno è quello di creare design che funzionino bene su dispositivi di diverse dimensioni. Per rendere questo più semplice, oltre alla possibilita di creare layout *fluidi*, cioè governati da misure percentuali che quindi scalano con la dimensione dello schermo, il CSS mette a disposizione le cosiddette **media query**. Una media query permette di applicare selettivamente alcune regole CSS solamente quando vengono soddisfatte determinate condizioni riguardo all'ambiente di visualizzazione della pagina, fra cui:

- `width`: la larghezza della finestra di visualizzazione;
- `height`: l'altezza della finestra di visualizzazione;
- `device-width`: la larghezza dello schermo del dispositivo;
- `device-height`: l'altezza dello schermo del dispositivo;
- `orientation`: l'orientamento dello schermo del dispositivo;
- `color`: il numero di bit di codifica dei colori.

Una media query si presenta in forma:

```
@media only screen and (max-width: 480px) {
  /* si applica a dispositivi con schermo di larghezza minore di 480 px */
}
```

dove il tag `only` indica di non applicare la regola su browser incompatibili con le media query. Alternativamente, si può scrivere:

```
@media all and (orientation: landscape) {
  /* si applica a dispositivi in orientamento landscape */
}
```

Attraverso una combinazione di *liquid design* e media query, si possono creare layout in cosiddetto *responsive design*, cioè che si adattano a diverse modalità di visualizzazione su diversi dispositivi.

6 Lezione del 21-10-24

6.1 JavaScript

Il linguaggio Javascript è fondamentale allo sviluppo web moderno. Il Javascript è un linguaggio di scripting (oggi perlopiù compilato JIT) orientato agli oggetti (attraverso il paradigma dei prototipi), a tipizzazione debole. Viene usato principalmente come linguaggio di scripting lato client, anche se alcuni framework permettono di usarlo come lingaggio lato server (Node.js).

Viene introdotto nel 1996 da Brendan Eich per Netscape Navigator (oggi Firefox). Nel 1997, l'ECMA standardizza l'ECMAScript, che è sià un sottoinsieme che un sovrainsieme del Javascript. L'ultima versione di rilievo in quanto a cambiamenti è l'ES6 (ha introdotto classi, iteratori e promesse), mentre la più recente è l'ES11 (o ES2020).

In Javascript una **variabile** è un'oggetto che può avere proprietà e metodi. Anche una funzione è considerata un'oggetto.

6.1.1 Scripting lato client

Per scripting lato client si intende codice che viene scaricato e quindi eseguito localmente sulla macchina client, e non sul server e restituito sotto forma di risposte HTTP.

- **Vantaggi:**
  - Elaborazione in locale significa meno carico sul server;
  - Il client può rispondere più velocemente all'utente di quanto potrebbe farlo un ciclo richiesta-risposta HTTP da parte del server;
  - Javascript può interagire direttamente con l'HTML della pagina, creando un'esperienza di sviluppo più simile a quella del software desktop.
- **Svantaggi:**
  - Non c'è garanzia che il client abbia abilitato Javascript, cioè ogni funzionalità assolutamente necssaria deve avere ridondanze lato server;
  - Le pagine che usano il Javascript in maniera massiccia sono difficili da mantenere;
  - Javascript non tollera errori. Se i file HTML e CSS possono essere invalidi, il browser annullerà l'esecuzione nel caso di errori del codice Javascript;
  - Javascript è correntemente supportato da quasi tutti i browser, ma gli standard e le API sono in continua evoluzione. Questo significa che le ultime funzionalità non potrebbero essere disponibili subito su tutti i browser.

6.2 Javascript e HTML

Il javascript può essere inserito in diversi modi nei nostri documenti HTML:

- **Inline:** si riferisce alla pratica di inserire del codice Javascript direttamente all'interno di attributi di elementi HTML. Ad esempio:

```
<a href="JavaScript:OpenWindow();">for more info</a>
<input type="button" onClick="alert('Are you sure?');"/>
```

- **Embedded:** contenuto all'interno dell'elemento `script`

```
<script>
  alert("Ciao vecchio!");
</script>
```

  Il codice inserito in modalità embedded viene eseguito al caricamento della pagina, tranne le funzioni che, ovviamente, vengono eseguite alla chiamata.

- **Esterno:** sempre indicato dall'elemento `script`, ma per riferimento attraverso l'attributo `src`, che si riferisce ad un file separato con estensione `.js`.

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

  All'interno dei file `.js` non si può includere alcun codice HTML (nemmeno i tag `<script>` / `</script>`).

L'esecuzione del codice avviene, come abbiamo detto, al caricamento della pagina per codice all'infuori di funzioni, e alla chiamata per codice all'interno di funzioni. Inoltre, vedremo come si possono eseguire funzioni in corrispondenza di determinati **eventi**.

Inoltre, conviene notare che quando incontra un file JavaScript, il browser arresta il rendering del documento HTML finché non ha finito di eseguire il codice ivi contenuto.

6.2.1 Utenti senza Javascript

Nel caso l'utente non voglia usare Javascript, si può usare l'elemento `noscript`, che viene abilitato solamente nel caso Javascript sia disattivato.

6.3 Variabili e tipi di dato

Una variabile in Javascript è debolmente tipizzata, ergo non bisogna dichiararne il tipo. Si usano le keyword `var`, `const` (per variabili costanti) o `let`. L'assegnamento può avvenire in qualsiasi momento co l'operatore `=`.

Esistono due tipi di dato:

- **tipi riferimento:** cioè gli oggetti. Contengono effettivamente riferimenti al blocco di dati che contiene l'oggetto;
- **tipi primitivi:** alcuni tipi speciali, che possono comunque essere usati come oggetti. Vengono solitamente immagazzinati direttamente in memoria. Questi sono:
  - **Booleani**, vero o falso;
  - **Numeri**, solitamente double precision su 64 bit;
  - **Stringhe**, di caratteri delimitate da `'` o `"`;
  - **Null**, il tipo nullo;
  - **Undefined**, il tipo non definito (non inizializzato);
  - **Simboli**, nuovo in ES2015, rappresenta un valore unico che può essere usato come chiave.

6.3.1 Oggetti predefiniti

Oltre ad array e funzioni, esistono altri oggetti predefiniti. Alcuni dei più usati sono Object, Function, Boolean, Error, Number, Math, Date, String e Regexp. Useremo alcuni oggetti vitali che non fanno parte della specifica Javascript, ma sono comunque supportati dal browser: questi sono window, console e document.

6.3.2 Concatenazione

Per concatenare le stringhe si usa l'operatore `+`:

```
const country = "France";
const city = "Paris";
let msg = city + " is the capital of " + country;
```

6.3.3 Condizionali

Esistono i condizionali a cui siamo abituati dal C++, delimitati da graffe. Notiamo, negli operatori di confronto, i due:

- `===`, uguaglianza **strict**, controlla uguaglianze di tipo e valore (quindi dà falso da tipi uguali in valore ma che richiedono conversioni);
- `!==`, diseguaglianza **strict**, come sopra ma con la diseguaglianza.

6.3.4 Vero e falso

In Javascript, ogni oggetto ha un valore `true` o `false` predefinito. Tutti i valori sono `true` tranne `Null`, `""`, `0`, `NaN` e `undefined`.

6.3.5 Gestione delle eccezioni

Sono previste gestioni delle eccezioni, con i blocchi try-catch del C++

```
try {
  funzione_maligna("Ciao");
}
catch(err) {
  alert("Error: " + err);
}
```

6.3.6 Array

Le arrey possono essere create come letterali, o attraverso la funzione costruttore:

```
const countries = [ "Canada", "France", "Italy" ];
const days = new Array("Lun", "Mar", "Mer", "Gio", "Ven", "Sab", "Dom");
```

Sono previsti anche i loop for-each:

```
for(let day of days) {
  // ...
}
```

Si possono **destrutturare** array attraverso l'operatore di accesso (`[0]`, ...), o in sequenza come:

```
primo = array[0];
secondo = array[1];
// oppure:
let [primo, secondo] = array;
```

6.3.7 Oggetti

Gli oggetti in Javascript vengono gestiti attraverso il paradigma a prototipi, cioè nuovi oggetti vengono creati non dalle classi, ma da altri oggetti. Un oggett è composto da un'insieme di coppie chiave-valore dette **properietà**.

Il modo più tipico di accedere ad un oggetto è attraverso la notazione letterale:

```
const oggetto = {
  chiave1: valore1,
  // ...
  chiaveN: valoreN
}
```

Si possono poi usare o l'operatore punto o le parentesi quadre:

```
oggetto.chiave1;
ogetto["chiave1"];
```

Le proprietà possono poi essere **annidate**:

```
const country1 = {
  name: "Canada",
  languages: { "English", "French" },
  capital: {
    name: "Ottawa",
    population: "400000"
  }
  // ...
}
```

con array o altri oggetti.

Si possono, come le array, **destrutturare** gli oggetti con la notazione comoposta:

```
let {id, title} = photo;
let {id, title, location:{country, city}}
```

6.3.8 Spread

Si può usare la sintassi di **spread** per copiare oggetti da un'array ad un altra. Ad esempio:

```
const foo = {name: client.name, country: photo.country};
```

6.3.9 Notazione JSON

JSON sta per Javascript Object Notation, ed è un modo, analogo all'XML, di rappresentare oggetti del Javascript: La differenza principale fra il Javascript e il JSON e che le proprietà sono racchiuse fra virgolette:

```
const obj = '{
  "name1": "value1",
  // ...
  "nameN": "valueN"
}';
```

Gli oggetti JSON vengono letti attraverso l'oggetto predefinito JSON:

```
text = '{
  "name1": "value1",
  // ...
  "nameN": "valueN"
}';
const obj = JSON.parse(text);
console.log(obj.name1);
```

Il JSON viene usato oggi per rappresentare dati strutturati anche al di fuori del web.

6.3.10 Funzioni come oggetti

Le funzioni in Javascript vengono definite con la parola chiave `function`, seguita dalla nome di funzione ed eventuali parametri. Come in Python, le funzioni non richiedono né un tipo di ritorno né i tipi degli argomenti. Vediamo un'esempio di **dichiarazione di funzione**:

```
function subtotal(price, quantity) {
  return price * quantity;
  //ci si aspetta che prenda due numeri e restituisca un'altro numero
}
```

Potremo invocare questa funzione come:

```
let result = subtotal(2, 3);
```

Si possono definire anche **funzioni espressione**, o *funzioni anonime*. Questo perchè, in Javascript, una funzione è essenzialmente un'oggetto:

```
const warn = function(msg) { alert(msg); };
warn("Questo non restituisce nulla");
```

Si possono specificare **valori di default**. Ad esempio:

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

let bar = foo(3); // bar e' NaN

function foo1(a = 10, b = 0) {
  return a + b;
}

let bar = foo1(3); //bar e' 3
```

In Javascript possono esistere funzioni con **numero variabile di argomenti**, usando la parola chiave `args` e l'operatore di **rest** (...). Ad esempio, si può avere:

```
function conc(...args) {
  let s = "";
  for(let a of args){
    s += a + " ";
    return s;
  }
}
```

Si possono poi avere **funzioni annidate**, quando abbiamo bisogno di funzioni dentro altre funzioni:

```
function calculateTotal(price, quantity) {
  let subtotal = price * quantity;
  return subtotal + calculateTax(subtotal);

  function calculateTax(subtotal) {
    let rate = 0.05;
    return subtotal * rate;
  }
}
```

Attraverso un procedimento di **hoisting**, la funzione `calculateTax` viene automaticamente spostata all'inizio del suo ambito di visibilità corrente (la funzione `calculateTotal`)

Possiamo quindi passare le stesse funzioni come argomenti ad altre funzioni, ovvero fornire delle cosiddette **funzioni callback** (*funzioni di "richiamo"*). Ad esempio:

```
const calculateTotal = function(price, quantity, tax) {
  subtotal = price * quantity:
  return subtotal + tax(subtotal)
}

const calcTax = function calculateTax() {
```

```
    let rate = 0.05;
    return subtotal * rate;
}

calculateTotal(2, 5, calcTax);
```

6.3.11 Metodi

Quelli che erano **metodi** negli altri linguaggi, non sono altro che proprietà funzione in Javascript. Un'oggetto può quindi avere funzioni definite come proprietà al suo interno. Si può usare, come in C++, la parola chiave `this` per riferirsi all'oggetto stesso.

Si possono avere **funzioni costruttrici**, cioè funzioni che si usano con la parola chiave `new` per creare nuovi oggetti. Queste usano, al loro interno, la parola chiave `this` per riferirsi all'oggetto creato.

```
function Customer(name, address) {
  this.name = name;
  this.address = address;
  // ...
}

let customer = new Customer("Marietto Stromboldi", "Via del Guzzuldo 56");
```

Quello che succede quando si esegue questo codice è che la parola chiave `new` crea un'oggetto vuoto, e la funzione costruttrice popola poi questo oggetti con gli attributi necessari.

6.3.12 Funzioni freccia

Le **funzioni freccia**, introdotte in ES6, forniscono una sintassi più concisa per le funzioni anonime. Forniscono anche un metodo per gestire gli errori di visibilità causati dalla parola chiave `this`. Per iniziare, consideriamo una semplice espressione di funzione:

```
const taxRate = function() { return 0.05; };
// questo puo' diventare
const taxRate = () => 0.05;
```

Esistono poi diverse varianti, che ci permettono di esprimere funzioni con livelli variabili di complessità:

Notiamo che nel caso precedente abbiamo usato l'ultima contrazione (senza argomenti).

Le funzioni freccia non forniscono un valore specifico a `this`, ma usano quello dell'ambito parentale corrente.

6.3.13 Visibilità

Esistono 4 livelli di visibilità (o *scopo*) in JavaScript:

- **Visibilità a livello funzione**: anche detta visibiltà locale, specifica a funzioni (ad esempio lo sono gli argomenti di funzione). `var`, `let` e `const` dichiarano variabili a livello visibiltà di funzione. Dobbiamo fare attenzione però, in quanto visibilità a livello funzione non significa visibilità a livello blocco, cioè quello che ci aspetteremo in linguaggi come il C++. Ad esempio, il codice:

Sintassi tradizionale	Sintassi freccia
`function() { // statement }`	`() => { // statement }`
`function(a, b) { // statement }`	`(a, b) => { // statement }`
`function() { call(); }`	`() => { call(); }`

`function(a) { return value; }`	`(a) => value;`
	 `// o addirittura`
	 `a => value;`

```
for (var i = 0; i < helpText.length; i++) {
  var item = helpText[i];
  document.getElementById(item.id).onfocus = function() {
    showHelp(item.help);
  }
}
```

ha un errore semantico, in quanto la variabile `item`, dichiarata come `var` in un blocco non funzione, è effettivamente visibile a livello funzione e va a sovrascriversi ad ogni esecuzione del ciclo for.

- **Visibilità a livello blocco**: incluso all'interno di un blocco di controllo. Dobbiamo stare attenti ad usare soltanto `let` o `const` quando si vogliono dichiarare variabili a visibilità a livello blocco, come appare chiaro dall'esempio precedente. La parola chiave `var` infatti, come abbiamo detto, dichiara variabili visibili a livello funzione o globale. Questa differenza deriva dal fatto che la visibilità a livello blocco, e quindi le relative parole chiave, sono state introdotte solo nella versione ES6 di JavaScript;
- **Visibilità di modulo:** riguarda il codice organizzato in file, o *moduli*, separati. Ogni variabile dicharata in un modulo è privata a quel modulo a meno che non venga **esportata** dal modulo e **importata** da un altro modulo;
- **Visibilità globale**: le variabili definite con visibilità globale (quindi abbiamo detto definite con `var` fuori dallo scopo) sono visibili da ogni parte del codice dopo la loro dichiarazione.

Si nota inoltre che il JavaScript supporta la dichiarazione implicita di variabili, che prendono automaticamente visbilità globale (come con `var`).

Notiamo che in JavaScript esite il meccanismo della cosiddetta **closure**: cioè, la chiusura di una funzione è un'oggetto che contiene la funzione e la sua catena di visibilità,

ergo che conserva il valore delle variabili a lei visibili a tempo di dichiarazione. Possiamo fare l'esempio:

```
function foo() {
  var name = "Pino";
  function saluta() {
    console.log("Ciao " + );
  }
  saluta();
}

/* ... */
foo();
```

Alla chiamata della funzione foo(), viene istanziata la variabile `name`, e poi viene chiamata la funzione `saluta()`. Questa non ha variabili locali, ma vede comunque la variabile `name` dichiarata dalla funzione madre. Ergo, la chiamata di `foo()` risulta nella corretta esecuzione di saluta(), cioè nell'output della stringa `"Ciao Pino"`, anche se effettivamente la funzione `saluta()` chiamata da `foo()` non ha una dichiarazione interna di `name`. Questo meccanismo prende anche il nome di **scoping lessicale**.

6.3.14 Output

Ci vengono fornite alcune funzioni di output predefinite:

- `alert()`: visualizza un messaggio di errore o allerta;
- `prompt()`: visualizza un messaggio con un campo di input di testo;
- `confirm()`: visualizza un messaggio con restituzione di un opzione si/no.

Alternativamente, si può usare:

- `document.write()`: scrive contenuti direttamente nel file HTML;
- `console.log()`: visualizza contenuti nella console Javascript del browser.

Vedremo in seguito metodi più sofisticati per interagire con la pagina e con l'utente.

7 Lezione del 28-10-24

7.1 Il modello a oggetti del documento

Il Document Object Model (DOM) è un meccanismo che ci permette di modificare dinamicamente i contenuti di una pagina web attraverso linguaggi di scripting come il JavaScript.

Si ha che, sebbene il codice HTML di una pagina sia effettivamente lineare, la struttura che esso va a definire è fondamentalmente **strutturata**: diversi elementi possono coesistera in relazioni di parentela padre/figlio, fratello/fratello, ecc... Questo significa che si può rappresentare un documento HTML attraverso una **struttura ad albero**, dove ogni elemento viene detto **nodo**.

Le interazioni col DOM avvengono attraverso un oggetto specializzato detto `NodeList` (letteralmente, "lista di nodi"), che mantiene appunto, in maniera simile ad un array, un insieme di nodi. A questo punto, quando si ha accesso al DOM della pagina, modificare un elemento significa trovare il nodo corrispondente in una NodeList, e modificarlo.

7.1.1 Tipi di nodo

Esistono, innanzitutto, alcuni tipi principali di nodi nel DOM:

- **Nodi elemento:** rappresentano elementi veri e propri del documento, come `div`, `a`, `p`, ecc...
- **Nodi testo:** rappresentano il testo contenuto in elementi come paragrafi, anchor, ecc...
- **Nodi attributo:** rappresentano attributi di elementi, come `class`, `id`, `src`, ecc...
- **Nodi commento:** rappresentano commenti nella struttura del documento HTML.

7.1.2 Proprietà fondamentali dei nodi

Un nodo ha, fra le altre, le seguenti proprietà essenziali:

- `childNodes`, una `NodeList` che contiene i figli del nodo;
- `firstChild`, il primo nodo figlio del nodo interessato;
- `lastChild`, come sopra, l'ultimo nodo figlio del nodo interessato;
- `nextSibling`, il prossimo nodo fratello del nodo corrente, cioè il nodo successivo sul livello corrente;
- `nodeName`, il nome del nodo, che equivale al nome del tag per i nodi tag, al nome dell'attributo per i nodi attributo, e a `#text` per i nodi testo;
- `nodeValue`, il valore del nodo, cioè Null per nodi elemento, il testo per nodi testo, e il contenuto dei commenti nei nodi commento;
- `parentNode`: il nodo genitore del nodo corrente;
- `previousSibling`, il precedente nodo fratello del nodo corrente, cioè il nodo precedente sul livello corrente;
- `textContent`, una rappresentazione testuale del nodo, da cui sono stati rimossi i tag.

Per quanto riguarda i nodi elemento, si hanno poi le proprietà:

- `classList`, una lista in sola lettura (con metodi get/set) delle classi CSS assegnate a questo elemento. Le classi, a differenza dello stile, sono da usarsi per la modifica a tempo di esecuzione degli stili, in quanto mantengono più isolate le implementazioni in CSS e in JavaScript, rispettivamente degli stili stessi e della logica che li governa;
- `className`, il nome della classe a cui appartiene l'elemento;
- `id`, l'id dell'elemento;
- `innerHtml`, l'HTML che descrive l'elemento, che può ma **non dovrebbe** essere modificato (vedi attacchi XSS);

- `style`, una lista delle proprietà CSS assegnate a questo elemento, che può influenzare effettivamente gli stessi parametri modificati attraverso un cambio di classe con `classList`, ma il cui utilizzo è sconsigliato in quanto si vanno a creare proprietà via scripting che non hanno riscontro nel CSS;
- `tagName`, il nome del tag che forma l'elemento.

Alcuni tag hanno poi proprietà aggiuntive:

- `href`, indica l'URL collegato nei tag `a`;
- `name`, identifica, in maniera simile a `id`, elementi. Riguarda principalmente campi dati per form: si può infatti usare per `a`, `input`, `textarea` e `form`;
- `src`, indica l'URL sorgente dei contenuti visualizzati da `img`, `input`, `iframe` e `script`;
- `value`, fornisce il valore di input dato dagli elementi di input, come `input`, `textarea` e `submit`.

7.1.3 Accedere al DOM

In JavaScript, si può ottenere un riferimento al DOM attraverso l'oggetto `document`. Questo, oltre al DOM, contiene anche proprietà riguardanti il documento stesso, fra cui l'URL, il tipo di encoding, ecc...

Esistono quindi 3 modi principali per modificare, o comunque interagire, con il DOM:

- **Metodi di selezione**: i metodi di selezione di `document` sono, in ordine di selettività decrescente, `getElementById(id)`, `getElementsByClassName(name)`, e `getElementsByTagName(name)`. Questi cercano, rispettivamente, i nodi per id, per nome di classe, o per nome di tag.

  Esistono poi altri metodi, fra cui ricordiamo `querySelector()` e `querySelectorAll`, che permettono di ottenere elementi nello stesso modo in cui lo faremmo in CSS, cio+ specificando una stringa di query. La differenza fra i due metodi è che `querySelector()` restituisce il primo elemento che rispetta la query, e `querySelectorAll()` li restituisce tutti;

- **Modificazione strutturale dei nodi**: esistono diversi metodi per modificare la struttura del DOM:
  - `appendChild`: inserisce un nuovo nodo figlio in fondo al nodo corrente;
  - `createAttribute`: crea un nuovo nodo attributo sul nodo corrente;
  - `createElement`: crea un nodo elemento;
  - `createTextNode`: crea un nodo di testo;
  - `insertAdjacentElement`: crea un nodo figlio in una di quattro posizioni relative al nodo corrente:
    * `beforebegin`: prima dell'elemento su cui si applica la funzione;
    * `afterbegin`: dentro l'elemento su cui si applica la funzione, come primo figlio;
    * `beforeend`: dentro l'elemento su cui si applica la funzione, come ultimo figlio;

    * `afterend`: dopo l'elemento su cui si applica la funzione.
  - `insertAdjacentText`: come sopra, ma per i nodi testo;
  - `insertBefore`: inserisce un nodo figlio prima di un nodo di riferimento nel nodo corrente;
  - `removeChild`: rimuove un nodo figlio dal nodo corrente;
  - `replaceChild`: rimpiazza un nodo figlio con un figlio diverso.

- **Metodi evento**: il DOM si interfaccia direttamente col gestore di eventi del JavaScript, attraverso il metodo `addEventListener()`. La prossima sezione si interessa appunto di quali funzionalità possono essere implementate attraverso questo sistema.

7.1.4 Temporizzazione del DOM

Il DOM viene caricato al momento di caricamento della pagina. Non dovremmo mai cercare di modificare il DOM prima che questo sia stato caricato, ergo bisogna controllare lo stato di caricamento del DOM prima di accedervi per modifiche. Questo può essere fatto attraverso, come vedremo, il sistema di **eventi** fornito dal JavaScript.

7.2 Gestione degli eventi

In JavaScript è supportato il meccanismo di gestione degli eventi attraverso **event handler**. Un event handler viene definito in codice dal programmatore, e poi registrato attraverso la funzione `addEventLister` di un elemento DOM. A questo punto, a tempo di esecuzione, quando l'evento verrà attivato si avrà che l'event handler verrà eseguito.UN esempio di questo sistema è il seguente:

```
function myHandler() {
  alert("Ahi!");
}

const btn = document.querySelector("#btn");
btn.addEventListener("click", myHandler);
```

In questo caso, al click del pulsante di classe `#btn`, la funzione `myHandler()` verrà invocata e visualizzerà un messaggio a schermo.

Spesso, è più conveniente usare funzioni anonime, o una arrow function, come event handler:

```
const btn = document.getElementById("btn");
btn.addEventListener("click", function() {
  alert("Sono una funzione anonima");
});

const btn = document.getElementById("btn");
btn.addEventListener("click", () => {
  alert("Sono una funzione freccia");
});
```

7.2.1 Tipi di evento

Esistono, oltre al `click`, diversi tipi di eventi. Questi sono:

- **Eventi di mouse**, fra cui:

  - `click`: visto finora, corrisponde al click del mouse su un elemento;
  - `dbclick`: corrisponde a un doppio click su un elemento;
  - `mousedown`: corrisponde alla prima pressione del mouse su un elemento;
  - `mouseup`: corrisponde al rilascio del mouse sull'elemento;
  - `mouseover`: corrisponde a passare (non necessariamente cliccare) con il mouse sopra un elemento;
  - `mouseout`: corrisponde a rimuovere il mouse da un elemento;
  - `mousemove`: corrisponde a muovere il mouse mentre si trova sopra un elemento.

- **Eventi di tastiera**, fra cui:
  - `keydown`: corrisponde alla pressione di un tasto da parte dell'utente;
  - `keyup`: corrisponde al rilascio di un tasto da parte dell'utente.

- **Eventi di tocco**, lanciati da interfacce touchscreen, simili agli eventi lanciati dal mouse;

- **Eventi form**, fra cui:
  - `blur`: lanciato quando quando un elemento di un form perde il focus;
  - `focus`: lanciato quando un elemento di un form ottiene il focus;
  - `change`: lanciato quando una casella di input cambia valore (magari per l'azione dell'utente);
  - `reset`: lanciato al reset del form;
  - `select`: lanciato quando si seleziona del testo;
  - `submit`: lanciato all'invio del form. In particolare, questo evento può essere usato per fare della prevalidazione dei contenuti prima di inviarli al server. Ad esempio:

```
document.querySelector("#loginForm").addEventListener("submit",
    function(e) {
  let pass = document.querySelector("#pw").value;
  if(pass="") {
    alert("Inserisci una password");
    e.preventDefault(); /* questo impedisce il submit chiamato
    dall'utente */
  }
});
```

  In generale, l'interazione e la validazione coi formi rimane uno degli usi più popolari del JavaScript. Possiamo infatti usare lo scripting per rispondere alle seguenti casistiche:

  - Movimento fra elementi, di cui possiamo approfittare per dare aiuto contestuale all'utente;
  - Cambiamento di dati di elementi di input, che possiamo usare per fare prevalidazione o come sopra, per dare aiuto contestuale;

  - Invio finale del form, prima del quale, come abbiamo visto, possiamo fare delle operazioni di prevalidazione. Bisogna notare che qualsiasi prevalidazione si faccia sul lato client andrà rifatta sul lato server, in quanto l'ingresso dal lato client non è mai sicuro (il codice inviato da server può essere stato alterato sulla macchina dell'utente). Fare queste validazioni sul lato client resta comunque utile per ridurre il traffico in uscita al server, in quanto in condizoni operative si possono notare errori prima dell'invio dei dati.

- **Eventi media**, relativi a contenuti multimediali, fra cui:
  - `ended`: lanciato quando dell'audio o del video ha finito di eseguire;
  - `pause`: lanciato quando dell'audio o del video è stato messo in pausa;
  - `play`: lanciato quando dell'audio o del video viene avviato;
  - `ratechange`: lanciato quando la velocità di playback di un contenuto multimediale cambia;
  - `volumechange`: lanciato quando il volume di un contenuto multimediale cambia.
- **Eventi frame**, fra cui:
  - `abort`: lanciato quando si arresta il caricamento di una risorsa;
  - `error`: lanciato quando il caricamento di una risorsa va in errore;
  - `load`: lanciato quando la finestra è stata caricata completamente;
  - `DOMContentLoaded`: lanciato quando il documento HTML è stato caricato e il DOM è pronto;
  - `orientationchange`: lanciato quando si cambia orientamento (dello schermo);
  - `resize`: lanciato quando si modifica la dimensione del documento;
  - `scroll`: lanciato quando si scrolla il documento;
  - `unload`: lanciato quando il documento viene rimosso dalla memoria, cioè quando l'utente lascia la pagina.

  Gli eventi frame permettono di implementare tecniche di **lazy loading**, o *caricamento pigro*, cioè di caricare contenuti non essenziali della pagina (script, immagini, contenuti multimediali) solo quando l'utente compie determinate azioni (come scorrere sulla pagina, spostarsi fra pagine, ecc...). Così facendo si riduce la cosiddetta **critical rendering path**, cioè il numero minimo di risorse che il browser deve scaricare prima di visualizzare la pagina, e quindi il tempo complessivo di accesso alla pagina.

7.2.2 Caricamento pagina e il DOM

Possiamo quindi risolvere introdotto due paragrafi fa, cioè quello di capire quando la pagina è caricata. Questo può essere verificato attraverso due eventi separati:

- `window.load`: questo evento viene lanciato quando l'intera pagina è caricata (incluse immagini, script e stylesheet). Spesso non è necessario aspettare tanto, in quanto si può semplicemente aspettare il prossimo evento:
- `document.DOMContentLoaded`: questo evento viene lanciato quando il documento HTML è stato completamente scaricato e interpretato. Basta per iniziare a fare modifiche scriptate sul DOM.

Possiamo quindi racchiudere tutto il nostro codice relativo al DOM all'interno di un handler per l'evento `DOMContentLoaded`, ad esempio come:

```
document.addEventListener("DOMContentLoaded", function() {
  /* il mio codice */
});
```

7.2.3 Oggetti evento

Ogni volta che viene lanciato un evento, si va a creare un oggetto relativo a quell'evento, che trasporta informazioni che possono essere raccolte attraverso le proprietà dell'oggetto stesso. Ad esempio, un'evento `click` contiene le proprietà `clientX` e `clientY`, che rappresentano la posizone del cursore dell'utente al momento del click.

Un'evento viene passato come oggetto alle funzioni event handler che prevedono un argomento. Ad esempio si può avere:

```
const menu = document.querySelectorAll("#menu li");
for(let item of menu) {
  item.addEventListener("click", menuHandler);
}
function menuHandler(e) {
  /* e contiene un oggetto dell'evento onclick */
  const x = e.clientX;
  const y = e.clientY;
  /* fai qualcosa con la posizione del cursore x, y */
}
```

7.3 Propagazione di eventi

Quando un'evento viene lanciato su un elemento che ha degli antenati, questo viene propagato sugli antenati. Per ragioni storiche, la propagazione avviene in due direzioni:

- **Fase di cattura, o *"trickle down"*:** nella fase iniziale, il browser propaga l'evento all'elemento più esterno (quindi il tag `html`), e poi via via nei tag interni fino a raggiungere il tag che ha lanciato l'evento in primo luogo. Il browser Netscape ha storicamente adottato questo modello;
- **Fase di bubbling:** nella seconda fase, l'evento viene propagato dall'elemento che lo ha lanciato fino agli elementi esterni. Il browser Explorer ha, di contro, adottato storicamente questo modello.

Oggi, i browser supportano entrambe le fasi, che solitamente svolgono esattamente in quest'ordine. In particolare, si può specificare alla chiamata di `addEventListener` quale fase si vuole usare, col valore default di `bubble`:

```
/* si crea un event listener per la funzione "funzioneCattura" in fase di
    cattura */
elem.addEventListener('click', funzioneCattura, true);
/* oppure */
elem.addEventListener('click', funzioneCattura, { capture: true; });
/* si crea un event listener per la funzione "funzioneBolleggia" in fase
    di bubbling */
elem.addEventListener('click', funzioneBolleggia, false);
/* oppure */
elem.addEventListener('click', funzioneBolleggia, {capture: false; });
/* oppure, visto che e' default */
elem.addEventListener('click', funzioneBolleggia);
```

7.3.1 Fermare la propagazione

Esiste un problema con questo tipo di propagazione degli eventi: a volte vorremmo che un evento venisse catturato, e che poi non si propagasse oltre nella fasse di bubbling. Per fare ciò, possiamo usare il metodo `stopPropagation()` in fase di bubbling, sull'oggetto evento ottenuto dall'event handler. Un'esempio è:

```
const btns = document.querySelectorAll(".plus");
for (let b of btns) {
  b.addEventListener("click", function (e) {
    e.stopPropagation();
    incrementCount(e);
  });
}
```

7.4 Delegazione di eventi

Il meccanismo di propagazione può semplificare la gestione degli eventi nel caso si abbiano più elementi che possono lanciare eventi che devono essere gestiti da un singolo event handler. In questo caso, anziché definire l'event handler su tutti gli elementi, come avevamo fatto nell'esempio precedente:

```
const btns = document.querySelectorAll(".plus");
/* li scorro tutti */
for (let b of btns) {
  b.addEventListener("click", function (e) { /* handler */ });
}
```

si può definire un singolo event handler su un elemento genitore a tutti gli elementi lanciatori di eventi. In questo caso, attraverso la propagazione in fase di bubbling, si ha che l'evento viene catturato da questo genitore indefferentemente da quale elemento sia a lanciarlo:

```
const parent = document.querySelector("#list");
parent.addEventListener("click", function (e) {
  if(e.target && e.target.nodeName == "BUTTON") {
    /* handler*/
  }
});
```

7.5 Dataset

Esiste una differenza di temporizzazione fra le variabili dichiarate nel codice HTML (e quindi ottenute dal DOM) e quelle dichiarate nel codice JavaScript di un event handler: a tempo di definizione, quest'ultimo non avrà accesso al DOM, e quindi a parametri specifici all'HTML. Per ovviare a questo problema si usano la proprietà `dataset`, e quindi gli attributi `dataset` nei tag HTML: questi sono particolari attributi, prefissi da `data-`, che permettono di inserire informazioni qualsiasi in un elemento HTML. Ad esempio, si potrà avere:

```
<img src="file.png" id="a", data-country="Mozambico">
```

in HTML, e:

```
const element = document.getElementById("a");
const country = element.dataset.country;
/* adesso hai "Mozambico" in country */
```

per accedervi in JavaScript.

8 Lezione del 18-11-24

8.1 Sviluppo lato server

Il prossimo linguaggio che vedremo, il **PHP** è un linguaggio di scripting **lato server**. Attraverso lo sviluppo lato server, non siamo più limitati a fornire pagine statiche atraverso il nostro web server, ma possiamo *generare* contenuti **dinamici**.

8.1.1 Alcune tecnologie lato server

Vediamo alcune tecnologie per lo sviluppo lato server:

- **JSP** (Java Server Pages): un ambiente che usa il linguaggio di programmazione Java. Veniva spesso usato nei sistemi informatici di grandi aziende (banche, ecc,,.). Ad oggi è stato soppiantato, al meno al di fuori degli ambiti aziendali, da altri framework;
- **Node.js**: un ambiente lato server che permette di usare il JavaScript anche sul lato server. Questo rende Node.js estremamente vantaggioso per gli sviluppatori che vogliono usare lo stesso linguaggio sia lato client che lato server. Inoltre, Node.js è il suo stesso web server, ergo elimina la necessità di altri web server (come ad esempio Apache);
- **PHP**: sta per *Personal Home Page*, o più recentemente *PHP: Hypertext Preprocessor*, è una tecnologia lato server che può essere usata direttamente all'interno dell'HTML. Viene compilato in una rappresentazione intermedia (*opcode*) simile al *bytecode* del Java. Viene usato largamente nei cosiddetti **CMS**, ossia Content Management System, fra cui ricordiamo WordPress.

8.2 Responsabilità del web server

Riassumendo, possiamo dire che il web server deve occuparsi di:

- Gestire connessioni attraverso il protocollo a livello di *application layer* HTTP;
- Rispondere alle richieste di risorse statiche (viste finora, in HTML, JavaScript e CSS) e dinamiche (generate col PHP o altre tecnologie lato server);
- Gestire i permessi all'accesso di determinate risorse;
- Cifrare e comprimere dati;
- Gestire più domini e URL;
- Gestire connessioni a tecnologie databse quali MySQL e MariaDB;
- Gestire cookie e stati;
- Caricare e gestire file generati dagli utenti o altri web server.

8.2.1 Lo stack LAMP

Abbiamo visto che tecnologie lato server vengono create attraverso determinati *software stack*, fra cui ricordiamo il LAMP (XAMPP su Windows), che comprende:

- **L**: il sistema operativo Linux (più probabilmente un sistema operativo GNU/Linux, costituito da software di GNU sul kernel Linux);
- **A**: il web server Apache;
- **M**: il DBMS (*Database Management System*) MariaDB (storicamente MySQL);
- **P**: il linguaggio di scripting PHP (e storicamente il *Perl*, che forma la seconda "P" in XAMPP).

Possiamo vedere Apache come un **intermediario** fra le richieste HTTP e il nostro codice in PHP. Quando il browser invia una richiesta, il web server carica la risorsa corrispondente, esegue l'eventuale codice PHP per generare risorse, ecc... e restituisce la pagina completa. Sui sistemi Linux, Apache è un **daemon** (*servizio* su Windows), cioè un processo che viene eseguito in background sulla macchina server.

Apache implementa le sue funzionalità attraverso diversi **moduli**:

- `mod_auth`: si occupa di autorizzare gli utenti che accedono alla pagina attraverso il protocollo HTTP;
- `mod_rewrite`: si occupa di accorciare gli URL e effettuare redirezioni;
- `mod_ssl`: si occupa della validazione di certificati SSL;
- `mod_php5`: PHP, a sua volta diviso in alcuni moduli:
  - **PHP core**: la libreria base del PH, che include funzionalità di base come gestione di stringhe, array, matematica, ecc...
  - **Extension layer**: agganci esterni per database, protocolli come l'FTP, e formati come l'XML;
  - **Zend Engine**: si occupa della lettura effettiva di un file PHP, la compilazione e l'esecuzione.

8.2.2 Apache e esecuzione parallela

Il server Apache può funzionare in due modalità fondamentali:

- **Multi-process**, anche detta *worker*: un nuovo processo viene creato per ogni richiesta che viene ricevuta;
- **Multi-threaded**, anche detta *preforked*: più processi (solitamente uno per thread, o *flusso di esecuzione*) gestiscono più richieste contemporaneamente.

8.3 PHP

Il PHP, come il JavaScript, ha sintassi *C-like* ("simile al C"). Gli script in PHP hanno estensione `.php`, e hanno l'aspetto di file HTML inframezzati da opportuni **tag PHP**:

```
1 <?php echo "Ciao vecchio!" ?>
```

8.3.1 Commenti

I commenti in PHP possono essere:

- Su linea singola, indicati con `//` o `#`;
- Su più linee, indicati con la classica sintassi `/* ... */`.

8.3.2 Variabili

Le variabili in PHP sono a **tipizzazione dinamica**, e il valore di una variabile può cambiare nel suo tempo di vita (*loose typing*). Per dichiarare (e in seguito accedere a) una variabile bisogna prefiggere al suo identificatore il simbolo `$`. I tipi supportati dal PHP sono:

- **Boolean**: tipi booleani (vero / falso);
- **Integer**: interi con segno;
- **Float**: numeri a virgola mobile;
- **String**: stringhe di caratteri;
- **Array**: collezioni di oggetti;
- **Object**: istanze di classe.

Le variabili **costanti** vengono definite attraverso la funzione `define()`, e di li in poi riferite senza usare il simbolo `$`.

8.3.3 Concatenazione di stringhe

Le stringhe in PHP vengono concatenate attraverso l'operatore punto (`.`). È permesso usare direttamente il nome di variabile di una stringa dentro a un altra stringase si antepone il prefisso `$`:

```
$firstName = "Tizio";
$lastName = "Caio";
// queste istruzioni sono equivalenti
echo "<em>" . $firstName . $secondName . "</em>";
echo "<em> $firstName $secondName </em>";
```

8.3.4 Sintassi alternativa per le strutture di controllo

Il PHP prevede una sintassi alternativa, oltre a quella tipica del C, per le varie strutture di controllo (if, for, ecc...). Si possono infatti usare i due punti al posto della graffa aperta (e quindi omettere la graffa chiusa):

```
<?php if($condizione = "vera") : ?>
  <a href="primo.php">Primo</a>
<?php else : ?>
  <a href="secondo.php">Secondo</a>
<?php endif ;?>
```

8.3.5 Output

L'output in PHP si può fare con la funzione (vista finora) `echo`, o attraverso la `printf`, derivata direttamente dal C. Ricordiamo che in C la `printf` si aspetta un **segnaposto**, in forma:

- `b`: binario;
- `d`: decimale (intero con segno);
- `f`: virgola mobile;
- `o`: ottale;
- `x`: esadecimale;
- `s`: stringa.

con eventuale specifica di precisione (ad esempio, `%.2f` restituisce un numero a virgola mobile con 2 cifre significative).

8.3.6 Inclusione di file

Il PHP permette di includere all'interno di script altri file (quindi altri script) in PHP. Questo si può fare attraverso le funzioni:

- `include`: include un file, e nel caso si verifichi un errore si limita a mostrare un avviso;
- `include_once`: come sopra, ma si assicura di non includere lo stesso file più volte;
- `require` :include un file, e nel caso si verifichi un errore arresta l'esecuzione;
- `require_once`: come sopra, come sopra;

L'inclusione avviene come una copiatura diretta del file dove si trova la direttiva di inclusione, e quindi allo stesso livello di visibilità (di blocco, di funzione, ecc...) in cui si trova la direttiva di inclusione stessa.

8.3.7 Funzioni

Le funzioni in PHP vengono dichiarate sempre attraverso la parola chiave `function`. Gli argomenti di funzione vengono detti in PHP **parametri**. È supportato il passaggio con valori di riferimento, e ancora per valore e per **riferimento**, usando la sintassi (`&`) del C. Si noti che il passaggio di default è comunque per **valore**.

Inoltre, notiamo che le variabili definite all'esterno di funzioni **non** sono visibili all'interno delle funzioni, a meno di non usare la parola chiave `global`:

```
$count = 56;

function func() {
  echo $count; // errore o output vuoto
  echo global $count: // tutto ok
}

echo $count; // tutto ok
```

8.3.8 Array in PHP

Le array in PHP sono effettivamente **mappe ordinate** (*array associativi*), cioè coppie ordinate di **indici** (sostanzialmente *chiavi*) e **valori**).

Possiamo infatti usare la sintassi tradizionale:

```
$days = array("Lun", "Mar", "Mer", "Gio", "Ven", "Sab", "Dom");
// oppure:
$days = ["Lun", "Mar", "Mer", "Gio", "Ven", "Sab", "Dom"];
```

o la sintassi associativa:

```
$days = array(0 => "Lun", 1 => "Mar", 2 => "Mer", 3 => "Gio", 4 => "Ven",
    5 => "Sab", 6 => "Dom");
```

Sono supportati, come in C, *array di array*, cioè **array multidimensionali**.

Per l'iterazione su array si può usare la funzione `count`, che restituisce la dimensione di un array, oppure il costrutto `foreach`, simle al `for(... of ...)` del JavaScript.

9 Lezione del 25-11-24

9.1 Superglobali

9.1.1 $_SERVER

`$_SERVER` rappresenta un'array associativa che contiene opzioni di configurazione per PHP e il server Apache, e gli header delle richieste HTTP inviate dai client.

Ad esempio, possiamo trovare le chiavi:

- `SERVER_NAME`: il nome del sito che è stato richiesto;
- `SERVER_ADDR`: l'indirizzo IP di tale server;
- `DOCUMENT_ROOT`: la directory da cui stiamo eseguendo lo script;
- `SCRIPT_NAME`: una chiave (il nome) dello script in esecuzione;
- `REQUEST_METHOD`: restituisce se il metodo di accesso con cui si è fatto accesso alla pagina è GET, HEAD, POST o PUT;
- `REMOTE_ADDR`: l'indirizzo IP del richiedente; curioso
- `HTTP_USER_AGENT`: sistema operativo e browser del client;
- `HTTP_REFERER`: l'indirizzo IP della pagina (se esiste) che conteneva il link usato per raggiungere la pagina corrente.

Ricordiamo che, come sempre, **non** possiamo fidarci del client, e quindi i valori di `HTTP_USER_AGENT` e `HTTP_REFERER` potrebbero essere falsificati.

9.2 Gestione di file caricati dall'utente

`$_FILES` rappresenta una variabile associativa che contiene oggetti che sono stati caricati dalla richiesta corrente, ciascuno come una coppia chiave-valore. Ricordiamo quindi che la trasmissione di file dal client si fa attraverso la richiesta POST. A questo punto, possiamo quindi creare un form che ottiene file dall'utente come segue:

```
<form enctype="multipart/form-data" method="post">
  <input type="file" name="file1" id="file1"/>
  <input type="submit"/>
</form>
```

Ogni elemento associato alla chiave per ogni file sarà un'array che conterrà i le chiavi:

- `name`: il nome del file sulla macchina del client;
- `type`: il tipo MIME (l'estensione) del file. Potremmo voler limitare i tipi di file supportati: questo si può fare lato server controllando l'estensione di ogni elemento di `$_FILES` caricato. Questo si può fare agevolmente con la funzione `explode()`, che segmenta la stringa in base a delimitatori specificati (nel nostro caso il punto "."), o direttamente controllando il campo `type` di ogni elemento di `$_FILES`.
- `tmp_name`: il nome del file sul server, che è una locazione temporanea;
- `error`: un intero che codifica diversi stati, fra cui ricordiamo UPLOAD_ERR_OK con valore 0 (che significa operazione andata a buon fine).
- `size`: un intero che rappresenta la dimensione in byte del file caricato. Potremmo voler limitare le dimensioni dei file inviati al nostro server. Possiamo fare ciò attraverso 3 meccanismi:
  - HTML nel form di input, cioè inserendo un elemento `input` nascosto con una coppia chiave valore `MAX_FILE_SIZE` e il valore in byte della dimensione massima dei file che il browser dovrà inviare. Questo meccanismo può essere manomesso dall'utente, e quindi va verificato nuovamente lato server. Notiamo inoltre che la maggior parte dei browser in commercio non supportano questo elemtno, ma è il PHP che solitamente si occupa di trasformare la sua inclusione in una legge che impedisce all'utente di caricare file più grandi del dovuto; chiarisci
  - JavaScript nel form di input, cioè controllando il file in uno script lato client. In particolare, l'elemento `input` di tipo `file` contiene un'array `files`, dove ogni file ha un campo `size`. Anche questo meccanismo è facilmente manomesso dall'utente.
  - PHP, controllando come nell'esempio precedente le dimensioni, ma stavolta lato server, cioè usando direttamente l'array `$_FILES` e il campo `size` dei suoi elementi,

9.2.1 Spostare i file

Possiamo spostare i file caricati dall'utente attraverso la funzione `move_uploaded_file()`, che prende come primo argomento la locazione temporanea `tmp_name` del file o altre locazioni dove esso si potrebbe trovare, e come secondo argomento l'indirizzo di destinazione nel filesystem del server.

9.3 Leggere e scrivere file

Ci sono due modi di leggere e scrivere file in PHP:

- **Accesso in stream:** il file viene letto una porzione alla volta, attraverso il concetto di stream, implementato analogamente ad altri linguaggi (come ad esempio il C).

- **Accesso all-in-memory:** il file viene caricato completamente in memoria, rendendo più facile la sua elaborazione.

Chiaramente, il primo approccio si presta a file di grandi dimensioni o a contesti dove le prestazioni sono fondamentali, mentre il secondo approccio è più agevole da usare, ma solo su file di piccole dimensioni.

9.3.1 Accesso in stream

Per aprire un file in modalità stream si usa la funzione `fopen()`. Da qui in poi il file è aperto come uno stream sequenziale, con un cursore che indica la posizione corrente, e sono disponibili le `fread()`, `fgets()`, `fwrite`, `fclose()`, ecc... a cui siamo abituati dal C.

9.3.2 Accesso all-in-memory

Nell'accesso all-in-memory, possiamo usare le funzioni `file()` che mette l'intero file in un array di stringhe rappresentanti ogni riga del file, `file_get_contents()`, che legge l'intero file in una variabile stringa, e `file_put_contents`, che scrive i contenuti di una variabile stringa in un file.

10 Lezione del 02-12-24

10.1 Connettere PHP a MySQL

Vediamo come connetterci, attraverso script PHP, al database MySQL (MariaDB su LAMP). Ciò che vogliamo fare è essenzialmente:

- Connetterci al DBMS;
- Gestire eventuali errori di connessione:
- Eseguire query sul database;
- Elaborare i risultati;
- Liberare le risorse e chiudere la connessione.

Esistono più librerie per l'interfacciamento con DBMS:

- **mysqli**, procedurale o orientata agli oggetti;
- **PDO**, orientata agli oggetti.

10.1.1 Connettersi al database

Vediamo come stabilire connessioni al database con mysqli e PDO:

- **mysqli**:

```php
// queste variabili variano di installazione in installazione
$host = "localhost";
$database = "testdb";
$user = "testuser";
$pass = "testpassword";

$connection = my_sqli_connect($host, $user, $pass, $database);
```

- **PDO**:

```
// queste variabili variano di installazione in installazione
$connection_string = "mysql:host=localhost;dbname=testdb";
$user = "testuser";
$pass = "testpassword";

$PDO = new PDO($connection_string, $user, $pass); // questa e' un'
    istanza di classe!
```

Notiamo che scrivere direttamente nel codice di connessione i dettagli del database e dell'utente che vi accede non è molto saggio. Un'opzione migliore è quella di definire variabili apposite:

```
<?php
define("DBHOST", "localhost");
define("DBNAME", "testdb");
define("DBUSER", "testuser");
define("DBPASS", "testpassword");
?>
```

10.1.2 Gestione degli errori

Le due librerie gestiscono diversamente gli errori di connessione:

- **mysqli** usa statement condizionali (if-else) sugli oggetti restituiti dai tentativi di connessione;
- **PDO** si basa su blocchi try-catch di gestione delle eccezioni.

10.1.3 Eseguire query

Vediamo come eseguire query negli oggetti restituiti da mysqli e PDO:

- **mysqli**:

```
$sql = "SELECT * FROM Biglietti WHERE Name = "Rossi"
$result = mysqli_query($connection, $sql);
```

- **PDO**:

```
$sql = "SELECT * FROM Biglietti WHERE Name = "Rossi"
$result = $PDO->query($sql);
```

10.1.4 SQL injection

Una grande falla di sicurezza che possiamo creare quando integriamo contenuti inseriti dall'utente con le nostre query è l'**SQL injection**. Se inseriamo ciò che otteniamo dall'utente così com'è nelle stringhe di query, questi potrebbe **chiudere** lo statement SQL corrente ed avviarne un altro, con conseguenze catastrofiche. Ad esempio, in una casella per l'ingresso dell'username, l'utente potrebbe scrivere:

```
"; TRUNCATE TABLE Users; #
```

A questo punto, se la nostra query aveva l'aspetto:

```
SELECT *
FROM Users
WHERE uname="$user_name"
  AND passwd=MD5("abcd")
```

otterremo la query "maligna":

```
SELECT *
FROM Users
WHERE uname="";
TRUNCATE TABLE Users; # AND passwd=MD5("abcd")
```

Questo comporterebbe l'eliminazione di tutti gli utenti, che chiaramente non è ciò che vogliamo.

Facciamo un'altra considerazione: nel codice presentato, della password viene memorizzato l'hash MD5, e non la password stessa *in chiaro*. Questa è sempre una misura di sicurezza, anche se oggi il semplice hashing MD5 non è più considerato un meccanismo di oscuramento delle password sicuro.

Possiamo proteggerci dall'SQL injection attraverso due meccanismi:

- **Sanitizzazione** dell'input dall'utente: questa si può fare attraverso le funzioni `mysqli_real_escape_string()` in mysqli o `quote()` in PDO;
- L'uso di **prepared statement**: un tipo di query che viene "precompilato" dal DBMS, e su cui la sanificazione viene eseguita automaticamente:
  - **mysqli**:

```
$id = $_GET["id"];
$sql = "SELECT * FROM Libri WHERE ID=?";

if($statement = mysqli_prepare($connection, $sql)) {
  // tipi di bind: s - stringa, b - blob, i - int, ecc...
  mysqli_stmt_bind_param($statement, "i", $id);
  mysqli_stmt_execute($statement);
}
```

  - **PDO**:

```
$id = $_GET["id"];

$sql = "SELECT * FROM Libri WHERE ID=?";
$statement = $PDO->prepare($sql);
$statement->bindValue(1, $id);
$statement->execute();
```

10.1.5 Elaborare i result set

Vediamo poi come elaborare i risultati ottenuti dalle query:

- **mysqli**: Abbiamo due alternative a seconda del tipo di query (normale o prepared):
  - **Query standard**:

```
$sql = "SELECT * FROM Iscrizioni WHERE YEAR(Data) > 2004";

if($result = mysqli_query($connection, $sql)) {
  while($row = mysqli_fetch_assoc($result)) {
    // $row contiene il record corrente
  }
}
```

  - **Query prepared**:

```php
$sql = "SELECT Nome, Data FROM Iscrizioni WHERE YEAR(Data) > 2004
    ";

if($statement = $mysqli_prepare($connection, $sql)) {
  mysqli_stmt_bindm($statement, "i", $id);
  mysqli_stmt_execute($statement);

  mysqli_stmt_bind_result($statement, $name, $data);

  while(mysqli_stmt_fetch($statement)) {
    // $name e $data contentgono i rispettivi attributi del
    record corrente
  }
}

```

- **PDO**:

```php
$sql = "SELECT Nome, Data FROM Iscrizioni WHERE YEAR(Data) > 2004";

$result = $pdo->query($sql);
while($row = $result->fetch()) {
  // $row contiene il record corrente
}
```

Notiamo che attraverso la libreria PDO è possibile estrarre gli attributi di un record in un oggetto anziché un array, a patto che i campi dell'oggetto corrispondano ai nomi degli attributi:

```php
class Libro {
  public $id;
  public $titolo;
  public $anno;
  public $descrizione;
}

// ...

$id = $_GET["ID"];
$sql = "SELECT id, titolo, anno, descrizione FROM Libro WHERE id=?"
$stmt = $pdo->prepare($sql);
$stmt->bindValue(1, $id);
$statement->execute();

$libro = $statement->fetchObject("Libro");
// adesso $book contiene gli attributi del record:
// $libro->id, $libro->titolo, ecc...
```

10.1.6 Chiudere la connessione

Vediamo infine come chiudere le connessioni col DBMS e quindi liberare le risorse relative:

- **mysqli**:

```php
$mysqli_free_result($result);
$mysqli_close($connection);
```

- **PDO**:

```
$pdo = null;
```

10.1.7 Transazioni

Sia mysqli che PDO supportano il meccanismo delle transazioni:

- **mysqli**:

```
mysqli_autocommit($connection, FALSE); // di default vale TRUE

// query

if($condition) {
  mysqli_commit($connection);
} else {
  mysqli_rollback($connection);
}
```

- **PDO**:

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO__ERRMODE_EXCEPTION); // per
    ottenere eccezioni dal DBMS

try {
  $pdo->beginTransaction();

  // query

  $pdo->commit();
} catch (Exception $e) {
  $pdo->rollback();
}
```

10.2 File nel database

Esistono due modi di immagazzinare file in un database:

1. Memorizzare la locazione del file nel filsystem del database;
2. Memorizzare il file direttamente nel database sotto forma di un **blob**. In particolare, vediamo un esempio di codice:

```
$fileContent = file_get_contents("file.rtf");
$sql = "INSERT INTO Documenti (File) VALUES(":data")";

$stmt = $pdo->prepare($sql);
$stmt->bindParam(":data", $fileContent, PDO::PARAM_LOB);
$stmt->execute();

// il file e' nel server, possiamo recuperarlo con:

$sql = "SELECT * FROM Documents WHERE id=:id";
$stmt = $pdo->prepare($sql);
$stmt->bindParam(":id", $_GET["id"]);
$stmt->execute();

$result = $stmt->fetch(PDO::FETCH_ASSOC);
if($result) {
```

```
17   // il file e' in $result["File"]
18 }
```

10.3 Gestione dello stato

Gestire lo stato nelle applicazioni web ha delle differenze sostanziali dal farlo nelle applicazioni desktop: mentre quest'ultime hanno a disposizione la memoria del dispositivo dell'utente, le applicazioni web dispongono solo della memoria del server, e quindi non hanno di norma modo di memorizzare informazioni relative alla sessione di un qualsiasi utente.

Possiamo introdurre meccanismi di recupero dello stato, in particolare:

- Stringhe di query: le stesse dei metodi POST e GET, ecc... poco usate per la gestione dello stato;
- **Cookie**.

10.3.1 Cookie

I cookie sono coppie nome-valore memorizzate in file di testo gestiti dal browser (con limite di dimensioni di 4 Kilobyte). Come sempre, visto che sono risorse generate sul lato client, non dobbiamo fidarci ciecamente di ciò che otteniamo dai cookie in quanto potrebbero essere stati manipolati.

Il principio di funzionamento dei cookie è il seguente: quando l'utente visita una pagina, il web server inserisce nell'header della risposta HTTP la rchiesta di impostare determinati cookie. Il browser, ricevute tali richieste, si occupa di impostare i cooke richiesti. Di lì in poi, ad ogni accesso al sito, il browser invierà (idealmente) i cookie che il web server aveva impostato indietro al web server stesso, così che questo possa recuperare informazioni sullo stato.

Le stringhe per l'impostazione di cookie nell'header hanno la forma seguente:

```
Set-Cookie name=value[;expires=date][;domain=domain][;path=path][;secure]
```

Il browser, di contro, invia indietro stringhe del tipo:

```
Cookie name=value
```

PHP fornisce la funzione `setcookie()` per impostare i cookie. Questa va eseguita prima di generare qualsiasi output HTML dallo script.

Vediamo le informazioni che può immagazzinare un cookie:

- **Scadenza** *(Expire)*: indica il tempo di validità del cookie, cioè il tempo per cui dovrebbe continuare ad essere inviato al web server per ogni richiesta HTTP. Un cookie con scadenza 0 è detto **cookie di sessione** e vale solo per la sessione corrente (sono di questo tipo i cookie inviati di default dalla `setcookie()`);
- **Dominio** *(Domain)*: indica il dominio per cui il cookie dovrebbe essere inviato, e può essere anche più grande del dominio corrente.
- **Percorso** *(Path)*: indica il percorso URL che deve esistere nella risorsa richiesta per l'invio del cookie;
- **Cookie sicuri** *(Secure)*: un flag che indica che l'invio del cookie deve accadere solo attraverso richieste SSL col protocollo HTTPS. I cookie inviati su HTTPS sono di default di questo tipo.

10.3.2 Serializzazione

La serializzazione è il processo attraverso il cui si trasformano oggetti arbitrariamente complicati in sequenze di *bit*, *byte*, o a livelli di astrazione superiori *caratteri* (solitamente in codifica ASCII o UTF-8).

PHP fornisce le funzioni `serialize()` e `unserialize()` proprio per questi scopi.

10.3.3 Gestione della sessione in PHP

PHP fornisce un meccanismo per la gestione della sessione. Questo si attiva attraverso la funzione `session_start()`. Dalla chiamata di questa funzione in poi, si può usare la variabile superglobale `$_SESSION` per accedere ad informazioni riguardo alla sessione.

A livello implementativo, il meccanismo di gestione della sessione si basa su un **cookie di sessione** da 32 bit che viene trasmesso avanti e indietro fra il client e il server. Al cookie sono associati dati serializzati (sostanzialmente array associativi), memorizzati nella memoria del server, che vengono restituiti da un componente detto **session state provider**.

Il problema può porsi quando si hanno più web server che si dividono il carico delle richieste: in questo caso i dati relativi ad una sessione potrebbero essere su uno solo di questi server, e un secondo non potrebbe proseguire correttamente la sessione di un utente. Le soluzioni al problema sono le seguenti:

- **Load balancer session-aware**: cioè load balancer che inviano richieste successive sempre allo stesso server in modo da preservare le sessioni in corso. Questo approccio rende il load balancer più complesso e meno efficiente.
- **Server di sessione condiviso**: un server a parte che si occupa di mantenere le informazioni riguardo a tutte le sessioni in corso.

10.3.4 Web storage

Il **web storage** è un meccanismo implementato lato client dal linguaggio JavaScript che permette la di supplementare i cokie attraverso dati memorizzati in locale sulla macchina del client.

Esistono due oggetti di web storage:

- **localStorage**: per informazioni persistenti fra sessioni;
- **sessionStorage**: per informazioni relative a singole sessioni.

Questi oggetti vengono usati come qualsiasi altro oggetto in JavaScript, cioè possono essere forniti di attributi arbitrari.